Андреа Далле Вакке

Zabbix. Практическое руководство

Andrea Dalle Vacche

Mastering Zabbix

Learn how to monitor your large IT environments using Zabbix with this one-stop, comprehensive guide to the Zabbix world

Second Edition

[PACKT] PUBLISHING
open source
community experience distilled
BIRMINGHAM – MUMBAI

Андреа Далле Вакке

Zabbix. Практическое руководство

Исчерпывающее руководство по организации мониторинга больших вычислительных окружений с использованием Zabbix

Второе издание

ДМК Пресс
издательство
Москва, 2017

УДК 004.7: 004.451.9Zabbix
ББК 32.972.5
Д15

Далле Вакке А.
Д15 Zabbix. Практическое руководство / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2017. – 356 с.: ил.

ISBN 978-5-97060-462-5

В книге описана система Zabbix – одно из самых популярных решений мониторинга сетей и приложений.

Описана настройка Zabbix, рассмотрены сценарии мониторинга, создание собственных компонент, автоматизация с использованием Zabbix API, а также интеграция Zabbix с внешними системами.

Издание предназначено системным администраторам и архитекторам, желающим интегрировать инфраструктуру Zabbix в свое окружение.

УДК 004.7: 004.451.9Zabbix
ББК 32.972.5

ISBN 978-1-78528-926-2 (анг.)
ISBN 978-5-97060-462-5 (рус.)

Содержание

Об авторе

Андреа Далле Вакке (Andrea Dalle Vacche)– высококвалифицированный профессионал с более чем 15-летним опытом работы в ИТ-индустрии.

Закончил университет в городе Феррара, Италия (Univeristà degli Studi di Ferrara) по курсу «Информационные технологии». Это образование послужило фундаментом, на который Андреа опирается в своих изысканиях с тех пор. Имеет сертификаты многих уважаемых крупных игроков в ИТ-индустрии, в том числе Cisco, Oracle, ITIL и, конечно, Zabbix. Также имеет сертификат «Red Hat Certified Engineer». На протяжении всей своей карьеры работал со многими масштабными вычислительными окружениями, часто на ролях, требующих обширных знаний. Это еще больше повысило его квалификацию, расширило круг практических навыков и укрепило стремление к применению полученных знаний на практике.

Любовь к Zabbix проснулась в Андреа, когда он занимался администрированием баз данных Oracle и разработкой приложений, использующих их. Основное время он тратил на снижение «затрат на эксплуатацию», специализируясь на мониторинге и автоматизации. Именно тогда он столкнулся с Zabbix и по достоинству оценил техническую гибкость инструмента и простоту администрирования с его применением. Используя новые знания в качестве стартовой площадки, Андреа вдохновился идеей создать Orabbix, первый комплект открытого программного обеспечения для мониторинга Oracle, который был бы полностью совместим с Zabbix. Он опубликовал несколько статей о программном обеспечении, связанном с Zabbix, таком как DBforBIX. Ознакомиться с его проектами можно на персональном веб-сайте автора: http://www.smartmarmot.com.

В настоящее время Андреа работает старшим архитектором в ведущем инвестиционном банке, в весьма разнородном вычислительном окружении. Он отвечает за очень широкий круг задач, сталкивается со многими критическими аспектами платформ Unix/Linux и вынужден особое внимание уделять разнородному стороннему программному обеспечению, имеющему стратегическую важность для развития банка.

Андреа также играет важную роль в группе обеспечения безопасности банка, занимаясь такими направлениями, как защищенность, сохранение тайны, стандартизация, аудит, удовлетворение требований регулятора и решения поддержки безопасности.

Кроме этой книги, он также написал:

- «Mastering Zabbix», Packt Publishing;
- «Zabbix Network Monitoring Essentials», Packt Publishing.

Благодарности

В первую очередь я хочу поблагодарить мою жену Анну (Anna) за ее поддержку и понимание. Она не раз оказывала мне помощь и давала ценные советы. Большое спасибо Фифи (Fifi) за умиротворяющее мурлыканье и пушистый покой.

Особое спасибо я хочу сказать всему коллективу издательства Packt Publishing и Адриану (Adrian) в частности. Их советы, поправки и предложения были по-настоящему ценными для меня. Весь коллектив проявил высокий профессионализм.

О технических обозревателях

Григорий Чернышев (Grigory Chernyshev) – старший инженер по выпуску/инженер по организации взаимодействий (senior release manager/DevOps engineer) в отделе онлайн-игр компании Mail.Ru Group. Специализируется на управлении конфигурациями, автоматизации процесса сборки, мониторинге, выпуске версий и создании сценариев на языке Python. Имеет опыт работы в таких проектах, как Allods Online и Skyforge – массовых ролевых игр AAA-класса, получивших известность по всему миру. В своей повседневной работе он использует Zabbix для мониторинга внутренних игровых серверов, гетерогенных агентов сборки и множества инфраструктурных серверов.

Помимо этого, он пишет плагины для систем Atlassian Jira и JetBrains Teamcity – в случае с последней даже победил на конкурсе WordPress Plugins в 2015 году!

Я хочу сказать спасибо моей жене за терпение, моим родителям за счастливое детство и координатору проекта, Санчите (Sanchita), за ее неиссякаемый энтузиазм и поддержку.

Нитиш Кумар (Nitish Kumar) – ведущий специалист по платформе Wintel в компании HT Media Ltd. и независимый технический блогер, занимающийся популяризацией самых разных технологий. Вот уже восемь лет занимается разными технологиями от Microsoft и открытыми решениями (включая, но не ограничиваясь: Spiceworks, продукты ManageEngine, Zabbix, MS Active Directory, MS Exchange Servers и др.), из которых два последних года посвятил рентабельным решениям корпоративного уровня с целью уменьшить сложность их требований и обеспечить более рациональное использование рабочего времени обслуживающего их персонала. Нитиш с большим энтузиазмом участвует в различных корпоративных событиях и общественных вебинарах. Особый интерес он испытывает к мобильным технологиям и часто пишет статьи о различных устройствах и технологиях. Имеет степень магистра информационных технологий, полученную в институте прикладной физики и технологий в Аллахабаде (Индия), и в область его интересов входят технологии от Microsoft, открытое программное обеспечение и мобильные устройства. Его блог находится по адресу: http://nitishkumar.net, желающие могут написать ему на электронный почтовый ящик: nitish@nitishkumar.net.

Нитиш является соавтором книги «Getting Started with Spiceworks», Packt Publishing. Также участвовал как технический обозреватель в подготовке других книг о Zabbix и Spiceworks.

Николас Пьер (Nicholas Pier) – сетевой инженер. Занимается веб-разработкой, проектированием сетевой инфраструктуры вычислительных центров на основе виртуализации и решений SAN. Пишет промежуточное программное обеспечение для бизнес-приложений. На даный момент Николас имеет множество промышленных сертификатов, включая Cisco CCNP, VMware VCP-DCV и множество других сертификатов от компаний Cisco и CompTIA. В свободное время увлекается пивоварением, бегом на длинные дистанции и чтением книг.

Тимоти Скоппетта (Timothy Scoppetta) – системный администратор. Специализируется на автоматизации, непрерывной интеграции и создании отказоустойчивых инфраструктур. Работал в Google и множестве начинающих компаний. В настоящее время занимается преподаванием ультрасовременных инструментов и эффективных приемов в институте.

Предисловие

С самого первого выпуска, состоявшегося в 2001 году, система Zabbix зарекомендовала себя как очень мощное и эффективное решение для мониторинга. Это открытый продукт, поэтому его легко получить и развернуть, а уникальный подход к мониторингу и отправке предупреждений позволяет на равных конкурировать с другими решениями, как открытыми, так и коммерческими. Это очень мощный и компактный пакет с очень низкими требованиями к аппаратуре и поддерживающему программному обеспечению. Если к перечисленному добавить еще простоту в использовании, становится очевидно, что Zabbix отлично подходит для мониторинга небольших окружений с ограниченным бюджетом. Но когда дело доходит до управления мониторингом большого количества объектов со сложными настройками и зависимостями, масштабируемость и распределенная архитектура Zabbix предстают в полном своем блеске. Как никакой другой продукт, Zabbix идеально подходит для больших и сложных распределенных окружений, позволяя эффективно управлять и извлекать полезную информацию из объектов мониторинга и событий, что особенно важно, если не важнее, чем решение обычных проблем доступности и простоты использования.

Это – второе издание книги, первое было написано в соавторстве с Андреа Далле Вакке (Andrea Dalle Vacche) и Стефано Кеван Ли (Stefano Kewan Lee).

Цель этой книги – помочь вам получить максимум от системы Zabbix и наладить эффективный мониторинг больших и сложных окружений.

О чем рассказывается в книге

Глава 1 «*Развертывание Zabbix*» описывает оптимальный выбор аппаратного и программного обеспечения для сервера Zabbix и базы данных с учетом текущей вычислительной инфраструктуры, целей мониторинга и возможного расширения в будущем. Эта глава включает также раздел с интереснейшим обсуждением размеров базы данных, который может пригодиться для оценки окончательного ее объема для стандартного окружения. Здесь также охватываются вопросы правильного определения размеров окружения и кратко обсуждаются измеряемые показатели, что также может пригодиться для планирования мощностей. Глава содержит практические примеры и теоретические расчеты, чтобы читатель мог получить навыки, необходимые для развертывания в действующем окружении.

Глава 2 «*Распределенный мониторинг*» исследует различные компоненты Zabbix, действующие на стороне сервера и клиента (агента). На одних и тех же примерах сетей будут даны различные распределенные решения, а также описаны их достоинства и недостатки. В дополнение к развертыванию и настройке аген-

тов здесь описываются настройки прокси-серверов, а также рассматриваются вопросы обслуживания и управления изменениями. В этом разделе охватываются все возможные архитектурные реализации Zabbix, а также положительные и отрицательные стороны.

Глава 3 «*Высокая доступность и отказоустойчивость*» охватывает вопросы высокой доступности и отказоустойчивости. Здесь вы научитесь выбирать параметры высокой доступности для каждого из трех основных уровней Zabbix. Обсуждение основывается на информации, представленной в двух предыдущих главах. Первая часть книги завершается несколькими сценариями развертывания, включающими высокодоступные серверы и базы данных, организованные в иерархические уровни и распределенные архитектуры, пригодные для мониторинга тысяч географически распределенных объектов. Эта глава включает практический пример и описание нескольких возможных сценариев.

Глава 4 «*Сбор данных*» выходит за рамки использования простых агентов и SNMP-запросов и затрагивает некоторые более сложные источники данных. В ней исследуются мощные встроенные функции Zabbix, порядок их использования и выбор параметров для мониторинга, чтобы обеспечить полный контроль без чрезмерного увеличения нагрузки на систему. Здесь также исследуются вопросы агрегирования значений и их использование в мониторинге сложных окружений с кластерами или еще более сложными грид-архитектурами (grid architectures).

Глава 5 «*Визуализация данных*» рассказывает о мощных возможностях визуализации данных в Zabbix. Она будет особенно полезна тем, кому требуется выяснить или обосновать необходимость расширения/обновления аппаратных средств. Здесь вы узнаете, как на основе данных мониторинга создавать динамические карты, организовать коллекции графиков для визуализации на больших экранах в центрах управления и реализовать общее качественное представление. Эта глава охватывает вопросы качественной визуализации результатов мониторинга, которая поможет своевременно выявлять проблемы и предупреждать их. Здесь также исследуются некоторые эффективные приемы использования отчетов о качестве обслуживания (Service Level Agreement, SLA), поддерживаемые системой Zabbix.

Глава 6 «*Управление оповещениями*» приводит примеры сложных триггеров и условий срабатывания, а также рекомендации по выбору правильного количества триггеров и оповещений. Ее цель – помочь выдержать баланс, чтобы не оставить незамеченными возможные проблемы и не вызвать появления большого числа ложных срабатываний. В этой главе вы также узнаете, как использовать действия для автоматического исправления простых проблем, активировать действия без участия человека с целью согласования разных триггеров и событий и внедрить их процесс управления. Кроме того, здесь вы узнаете, какие операции можно автоматизировать, чтобы уменьшить нагрузку на администраторов и оптимизировать процесс администрирования, дополнив его возможностью опережения событий.

Глава 7 «*Управление шаблонами*» содержит рекомендации по эффективному управлению шаблонами: конструирование сложных шаблонов из простых ком-

понентов, управление эффектами, вызванными изменениями в шаблонах, поддержка существующих объектов мониторинга и связывание шаблонов с вновь обнаруженными узлами сети. Эта глава завершает вторую часть книги, посвященную различным средствам мониторинга и управления данными, имеющимися в Zabbix. В третьей, заключительной части книги обсуждаются возможности интеграции Zabbix со сторонними продуктами и мощные возможности расширения системы.

Глава 8 «Внешние сценарии» рассказывает, как писать сценарии для мониторинга объектов, которые не поддерживаются ядром Zabbix. Описывает преимущества и недостатки хранения сценариев на стороне сервера или агента, как запускать или планировать их выполнение, и подробно анализирует протокол агентов Zabbix. Здесь вы узнаете обо всех возможных побочных эффектах, задержках и нагрузке, вызванных сценариями; научитесь реализовывать все необходимые проверки, зная все, что связано с ними. Эта глава включает примеры различных сценариев на Bash, Java и Python, опираясь на которые, вы легко сможете написать свои сценарии, расширяющие возможности мониторинга Zabbix.

Глава 9 «Расширение Zabbix» углубляется в прикладной интерфейс Zabbix и особенности его использования для создания специализированных интерфейсных элементов и сложных расширений. Она охватывает также вопросы выборки результатов мониторинга для дальнейшего исследования и составления отчетов. Включает простые примеры на Python реализации экспортирования и дальнейшей обработки данных, выполнения массовых и сложных операций, связанных с мониторингом объектов, и, наконец, автоматизации различных аспектов управления, таких как создание и настройка учетных записей пользователей, их активация и т. п.

Глава 10 «Интеграция с Zabbix» завершает книгу обсуждением вопросов интеграции Zabbix с другими системами. Интеграция имеет большое значение для успешного управления любыми большими и сложными окружениями. Здесь вы узнаете, как использовать встроенные особенности Zabbix, обращаться к прикладному интерфейсу или напрямую к базе данных для обмена информацией с различными выше- и нижестоящими системами и приложениями. Познакомитесь с конкретными примерами организации взаимодействий с приложениями инвентаризации, системами паспортизации отказов и системами хранения данных.

Кому адресована эта книга

Как следует из названия книги – «Zabbix. Практическое руководство. Второе издание», вы не найдете здесь подробных, пошаговых инструкций (за исключением, может быть, описания процедуры установки, которая будет описана с самого начала) по основам использования Zabbix. Несмотря на то что в книге приводится масса подробной информации по установке сервера или настройке элементов, триггеров и экранов, в ней предполагается, что вы уже имеете некоторое знакомство с особенностями работы системы и готовы сосредоточить внимание на более про-

двинутых аспектах. Даже если прежде вам не приходилось использовать Zabbix, вы все равно сможете почерпнуть немало ценного из этой книги, но в этом случае я настоятельно рекомендую обратиться к официальной документации Zabbix, доступной по адресу: https://www.zabbix.com/documentation/2.4/ru/manual, чтобы восполнить любые пробелы в ваших знаниях.

Что потребуется для работы с книгой

Прежде чем углубиться в настройки Zabbix, хочется отметить, что конфигурация, предлагаемая и обсуждаемая здесь, протестирована в крупном промышленном окружении (начитывающем более 1800 сетевых узлов, 89 500 элементов мониторинга и 30 000 триггеров), достаточно представительном для многих больших и очень больших окружений. Решения по поддержке высокой доступности, демонстрирующиеся в этой книге, являются не просто умозрительными рекомендациями, но были проверены на практике в ходе случившейся аварии (когда сетевые кабели были повреждены во время земляных работ).

Надо понимать, что большинство вариантов выбора из представленных в этой книге было сделано и проверено на практике. Одним из важнейших примеров может служить выбор PostgreSQL в качестве официальной СУБД для Zabbix. Система управления базами данных PostgreSQL – достаточно зрелая для промышленного использования и обладает очень богатыми функциональными возможностями:

- горячее резервирование поддерживается изначально;
- полноценная поддержка требований ACID (Atomicity, Consistency, Isolation, Durability – Атомарность, Согласованность, Изолированность, Долговечность) к транзакционной системе;
- множество различных встроенных способов организации резервных баз данных (горячее резервирование, синхронная репликация и т. д.);
- эффективное секционирование.

База данных является критически важным компонентом для Zabbix, особенно когда требуется хранить исторические данные и гарантировать постоянную производительность с ростом объема базы данных.

В этой книге мы будем исходить из нескольких предположений: в качестве системы управления пакетами используется `yum`, а операционная система установлена из дистрибутива Red Hat Enterprise Linux. Но в любом случае, кроме конкретных названий пакетов и диспетчера управления пакетами, информация в книге остается действительной для любых дистрибутивов Linux. Кроме того, обсуждаемые архитектуры и их реализации не связаны с каким-то определенным дистрибутивом. Мы не будем использовать оригинальную поддержку кластеров в Red Hat, точно так же мы не будем принимать решения, которые нельзя было бы воплотить в любом другом дистрибутиве Linux.

В книге часто встречаются упоминания различных открытых программных продуктов, но из всех них вам желательно иметь знакомство со следующими:

- Apache: http://www.apache.org/;
- Pacemaker: http://clusterlabs.org/;
- PostgreSQL: http://www.postgresql.org/;
- DRBD: http://www.drbd.org.

В этой книге также предполагается, что вы имеете некоторые навыки системного администрирования и программирования. Мы будем время от времени давать задания для самостоятельной реализации программного кода. Имея перед глазами предлагаемые примеры с подробным описанием, вы наверняка справитесь с созданием собственных плагинов или внешних программ, хорошо интегрирующихся с Zabbix. Примеры программного кода в книге написаны на двух широко распространенных языках: Java и Python. Они знакомы большинству современных программистов, а после знакомства с особенностями реализации протокола Zabbix вы без труда сможете переключаться между ними.

Zabbix – это не просто программный продукт для мониторинга; это открытое решение мониторинга, удовлетворяющее самые широкие потребности, и эта книга познакомит вас со всеми достоинствами и недостатками возможных решений.

А теперь пришло время вступить в мир Zabbix!

Соглашения

В этой книге вы обнаружите несколько стилей оформления текста, которые разделяют различные виды информации. Ниже приводятся примеры этих стилей и поясняется их значение.

Элементы программного кода в тексте, имена таблиц в базах данных, имена папок и файлов, расширения файлов, пути к каталогам в файловой системе, фиктивные адреса URL, ввод пользователя и учетные записи в Twitter оформляются так: «Большинство из этих параметров определяется в файле `php.ini`».

Блоки кода оформляются следующим образом:

```
zabbixsrv=zabbixsvr
[ -e /etc/sysconfig/$syscf ] && . /etc/sysconfig/$syscf

start()
{
    echo -n $"Starting Zabbix server: "
```

Когда потребуется привлечь ваше внимание к определенному фрагменту в блоке программного кода, он будет выделяться жирным:

```
; Максимальный объем данных в запросах POST,
; которые может обрабатывать PHP.
; http://www.php.net/manual/en/ini.core.php#ini.post-max-size
post_max_size = 16M
```

Ввод или вывод в командной строке будет оформляться так:

```
# yum list postgres*
```

Новые термины и **важные** слова будут выделены жирным. Текст, отображаемый на экране, например в меню или в диалогах, будет оформляться так: «Закончив заполнение формы, щелкните на кнопке **Next**».

Так оформляются предупреждения и важные примечания.

Так оформляются советы и рекомендации.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф в разделе «Читателям – Файлы к книгам».

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты автор-

ских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Вопросы

Вы можете присылать любые вопросы, касающиеся данной книги, по адресу dmkpress@gmail.com или questions@packtpub.com. Мы постараемся разрешить возникшие проблемы.

Глава 1

Развертывание Zabbix

Если вы читаете эту книгу, значит, вы почти наверняка уже устанавливали и использовали Zabbix. Почти наверняка вы пытались использовать эту систему в небольшом или среднем окружении, но с тех пор ситуация изменилась, ваше окружение разрослось, и вы столкнулись с новыми проблемами. В наши дни окружения растут или изменяются очень быстро, и порой довольно сложно оставаться в полной готовности поддерживать надежный мониторинг.

Обычно начальное развертывание системы мониторинга выполняется под руководством какого-либо самоучителя или инструкции, и это распространенная ошибка. Такой подход оправдан для небольших окружений, где время простоя не имеет большого значения, где не приходится беспокоиться о проблемах восстановления сайтов после аварий и, вообще, где все выглядит очень просто.

Почти всегда в таких случаях развертывание и настройка выполняются без учета появления новых элементов, триггеров и событий, которые должны обрабатываться сервером. Если вы уже установили Zabbix и желаете реализовать возможность дальнейшего расширения своего решения мониторинга или, напротив, решили спроектировать и создать новую инфраструктуру мониторинга, эта глава поможет вам.

Данная глава поможет также решить сложную задачу настройки/обновления Zabbix для использования в больших и очень больших окружениях. Она охватывает все аспекты такой задачи, начиная от определения большого окружения до использования Zabbix в роли ресурса с планируемой мощностью. Здесь будут представлены все возможные решения на основе Zabbix, включая практический пример установки с прицелом на обслуживание большого окружения и возможность дальнейшего совершенствования.

К концу этой главы вы узнаете, как действует Zabbix, какие таблицы требуют особого внимания, как рационализировать административные работы в большом окружении, что, как показывает опыт прошлых лет, является действительно очень сложной задачей.

В этой главе рассматриваются следующие темы:

- как определить, что перед вами действительно большое окружение, и какие окружения можно считать большими;
- настройка/обновление Zabbix в большом и очень большом окружении;

- установка Zabbix в трехуровневой системе при наличии готового решения для большого окружения;
- оценка требований к базе данных и определение общего объема хранимых данных;
- знакомство с наиболее тяжелыми таблицами и задачами базы данных;
- оптимизация работ с целью снижения нагрузки на СУБД и повышения эффективности системы в целом;
- основные идеи планирования мощности с учетом возможностей Zabbix.

Определение размера окружения

Основное внимание в этой книге уделяется большим окружениям, поэтому нужно определить, хотя бы приблизительно, какое окружение можно считать большим. Размер определяется разными характеристиками, но в самом простом случае окружение можно назвать большим, если:

- оно распределено географически;
- количество контролируемых устройств исчисляется сотнями или даже тысячами;
- количество проверок, выполняемых каждую секунду, превышает 500;
- имеется большое количество элементов, триггеров и данных для обработки (объем базы данных превышает 100 ГБ);
- доступность и производительность являются критически важными характеристиками.

Все эти пункты характеризуют большое окружение; установка и обслуживание инфраструктуры Zabbix в таком окружении играют важную роль.

Установка является четко определенной задачей, для выполнения которой специально выделяется время, и относится к разряду важнейших, потому что создает основу для мощной и надежной инфраструктуры мониторинга. Кроме того, после создания некоторого задела порой очень непросто что-то передвинуть/переместить без потери данных. Существуют и другие аспекты, которые необходимо учитывать: у нас появится множество задач, связанных с системой мониторинга, большинство из которых придется решать ежедневно, но в больших окружениях они требуют особого внимания.

В небольших окружениях с маленькими базами данных резервное копирование требует минутных усилий, но для большой базы данных решение той же задачи займет намного больше времени.

Планы по восстановлению должны регулярно пересматриваться и тестироваться, чтобы знать, сколько времени потребуется на решение этой задачи в случае аварийных ситуаций.

Помимо повседневного обслуживания, необходимо предусмотреть время на тестирование и ввод в эксплуатацию обновлений, чтобы максимально уменьшить их влияние на решение повседневных задач.

Архитектура Zabbix

Zabbix можно определить как распределенную систему мониторинга с централизованным веб-интерфейсом (с помощью которого осуществляется управление). Из основных особенностей системы хотелось бы особо выделить следующие:

- Zabbix имеет централизованный веб-интерфейс;
- сервер может выполняться под управлением практически любой Unix-подобной операционной системы;
- данная система мониторинга имеет готовых агентов для большинства операционных систем Unix, Unix-подобных и Microsoft Windows;
- система легко интегрируется с другими системами благодаря прикладному интерфейсу, реализованному для множества языков программирования;
- мониторинг может осуществляться по протоколам SNMP (v1, v2 и v3), IPMI, JMX, ODBC, SSH, HTTP(S), TCP/UDP и Telnet;
- данная система мониторинга позволяет нам создавать собственные элементы и графики и интерполировать данные;
- простота настройки.

На рис. 1.1 изображена трехуровневая архитектура Zabbix.

Рис. 1.1 ❖ Трехуровневая архитектура Zabbix

Архитектура Zabbix для большого окружения состоит из трех разных серверов/компонентов (которые также должны настраиваться с учетом высокой доступности):

- веб-сервер;
- сервер баз данных;
- сервер Zabbix.

Полная инфраструктура Zabbix в больших окружениях позволяет вводить в игру еще двух актеров, играющих важную роль. Это агенты Zabbix и прокси-серверы Zabbix. Пример такой инфраструктуры представлен на рис. 1.2.

В этой инфраструктуре имеется централизованный сервер Zabbix, к которому подключено несколько прокси-серверов, обычно по одному на ферму или подсеть.

Сервер Zabbix получает данные от **прокси-серверов Zabbix**, прокси-серверы получают данные от подключенных к ним **агентов Zabbix**, все данные сохраняются в выделенной СУБД, а доступ к этим данным осуществляется посредством веб-интерфейса. Если заглянуть в реализацию системы, можно увидеть, что веб-интерфейс написан на языке PHP, а сервер, прокси-серверы и агенты – на языке C.

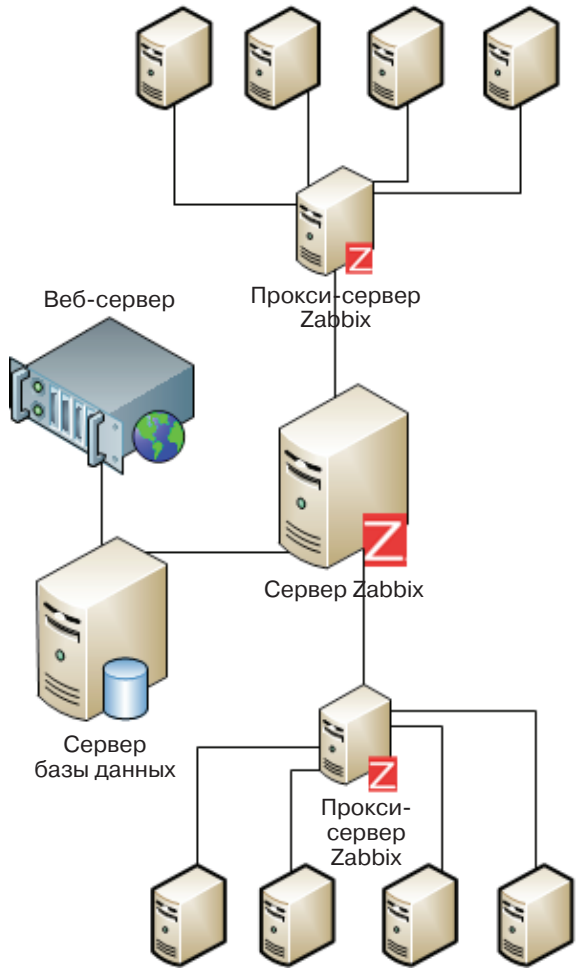


Рис. 1.2 ❖ Два дополнительных компонента инфраструктуры Zabbix

Выбор языка C для реализации сервера, прокси-серверов и агентов обусловлен стремлением обеспечить наивысшую производительность и минимальное потребление системных ресурсов. Все компоненты оптимизированы на достижение максимальной производительности.

Используя прокси-серверы, можно реализовать разные архитектуры. Ниже перечислено несколько таких архитектур в порядке возрастания сложности:

- с единственным сервером;
- с единственным сервером и множеством прокси-серверов;
- распределенная архитектура (доступна только в версиях Zabbix 2.3.0 или выше).

Архитектура с единственным сервером не предполагает использования в большом окружении. Это простейшая архитектура, где мониторинг осуществляет единственный сервер, которая может служить неплохой начальной точкой.

Вероятнее всего, у вас уже имеется установленная система Zabbix. Zabbix отличается высокой гибкостью и позволяет расширить установку с единственным сервером до следующего уровня: мониторинга с применением прокси-серверов.

Мониторинг с применением прокси-серверов реализуется путем настройки одного сервера Zabbix и нескольких прокси-серверов, по одному на ветвь или вычислительный центр. Такая конфигурация проста в обслуживании и обладает преимуществом решений централизованного мониторинга. Она обеспечивает хороший баланс между сложностью реализации и мониторингом большого окружения. Архитектуру с единственным сервером и множеством прокси-серверов, изображенную на рис. 1.2, можно (приложив немало усилий) расширить до распределенной архитектуры.

Начиная с версии 2.4.0 система Zabbix не поддерживает сценариев распределенного мониторинга узлов. И действительно, если загрузить исходный код версии Zabbix, обсуждаемой в книге, а затем код версии Zabbix 2.4.3, можно заметить, что ветвь кода, управлявшая узлами, была удалена.

Все возможные архитектуры Zabbix подробно обсуждаются в *главе 2 «Распределенный мониторинг»*.

Установка Zabbix

Конфигурация, обсуждаемая в этой главе, предусматривает создание отдельного сервера для каждого из следующих основных компонентов:

- веб-интерфейс;
- сервер Zabbix;
- база данных Zabbix.

Мы опишем эту конфигурацию потому, что:

- она может служить основой для дальнейшего расширения за счет добавления прокси-серверов и узлов;
- каждый компонент выполняется на выделенном сервере;
- подобные конфигурации являются отправной точкой для организации мониторинга больших окружений;
- она имеет большое распространение;
- она почти наверняка станет для вас отправной точкой, пригодной для дальнейшего расширения и совершенствования инфраструктуры мониторинга.

Фактически эта первая установка пригодится всем, кто собирается расширить существующую инфраструктуру. Если имеющееся у вас решение для мониторинга реализовано как-то иначе, вам стоит запланировать переход на архитектуру с тремя выделенными серверами.

Если после организации трехуровневой системы производительность все еще оставляет желать лучшего, можно запланировать и обдумать конфигурацию, лучше подходящую для ваших условий.

Осуществляя мониторинг большого окружения, следует:

- использовать выделенные серверы, чтобы упростить дальнейшее расширение;
- реализовать конфигурацию с высокой доступностью;
- реализовать конфигурацию с высокой отказоустойчивостью.

В трехуровневой системе нагрузка на центральный процессор серверного компонента не имеет большого значения, по крайней мере для сервера Zabbix. Потребляемая вычислительная мощность напрямую зависит от количества хранимых элементов и частоты обновления (количества проверок в минуту), а не от объема памяти.

В действительности сервер Zabbix не особенно требователен к производительности центрального процессора, но весьма взыскателен к объему памяти. По опыту, четырехъядерного процессора с 8 ГБ памяти вполне достаточно для мониторинга более чем 1000 хостов.

Существуют два основных способа установки Zabbix:

- загрузить последнюю версию исходного кода и скомпилировать его;
- установить из предварительно скомпилированного пакета.

Существует еще один путь: загрузить образ виртуальной машины с сервером Zabbix, настроить его и запустить, но мы не будем рассматривать этот способ, потому что полный контроль и знание необходимых шагов намного лучше. Кроме того, главный недостаток такого решения – в том, что эти образы (доступны по адресу: http://www.zabbix.com/ru/download.php), по утверждению самой компании Zabbix, непригодны для промышленной эксплуатации.

Установка из предварительно скомпилированных пакетов имеет следующие преимущества:

- упрощает процедуру обновления;
- автоматически удовлетворяет зависимости.

Установка из исходных кодов также имеет свои преимущества:

- возможность компиляции только необходимых составляющих;
- возможность статической сборки агента и установки в разных версиях Linux;
- возможность полного контроля над обновлениями.

Большие окружения обычно включают компьютеры, работающие под управлением разных версий Linux, Unix и Microsoft Windows. Такие сценарии весьма типичны в гетерогенных инфраструктурах, и если необходимо установить агента Zabbix на каждый Linux-сервер, придется иметь дело с разными версиями агентов и конфигурационными файлами, размещенными в разных местах.

Чем выше уровень стандартизации на серверах, тем проще обслуживать и обновлять инфраструктуру. Флаг компиляции `--enable-static` дает возможность стандартизовать агента для разных версий и выпусков Linux, и это является большим преимуществом. Агента, скомпилированного статически, легко можно развернуть где угодно и использовать один и тот же конфигурационный файл, хранящийся в одном месте. Процедуру развертывания также можно стандарти-

зовать; единственное, что может отличаться, – это сценарий запуска/остановки и порядок регистрации на требуемом уровне запуска.

Та же идея применима к коммерческим версиям Unix – того же самого агента можно устанавливать на разные версии Unix, выпускаемые одним поставщиком.

Предварительные требования

Перед компиляцией Zabbix необходимо ознакомиться с предварительными требованиями. Для поддержки веб-интерфейса нужно установить следующее программное обеспечение:

- Apache (1.3.12 или выше);
- PHP (5.3.0 или выше).

Для сборки сервера Zabbix необходимы:

- СУБД: можно использовать открытые альтернативы, такие как PostgreSQL и MySQL;
- пакет `zlib-devel`;
- пакет `mysql-devel`: реализует поддержку MySQL (не требуется в нашем примере);
- пакет `postgresql-devel`: реализует поддержку PostgreSQL;
- пакет `glibc-devel`;
- пакет `curl-devel`: используется для поддержки веб-мониторинга;
- пакет `libidn-devel`: требуется для пакета `curl-devel`;
- пакет `openssl-devel`: требуется для пакета `curl-devel`;
- пакет `net-snmp-devel`: реализует поддержку SNMP;
- пакет `popt-devel`: может требоваться пакету `net-snmp-devel`;
- пакет `rpm-devel`: может требоваться пакету `net-snmp-devel`;
- пакет `OpenIPMI-devel`: реализует поддержку IPMI;
- пакет `iksemel-devel`: реализует поддержку протокола Jabber;
- пакет `Libssh2-devel`;
- пакет `sqlite3`: необходим для поддержки SQLite, используется как промежуточная база данных Zabbix (обычно на прокси-серверах).

Установить все зависимости в Red Hat Enterprise Linux можно с помощью `yum` (с привилегиями root), но сначала нужно подключить репозиторий EPEL следующей командой:

```
# yum install epel-release
```

Выполните команду `yum install`, чтобы установить все необходимые пакеты:

```
# yum install zlib-devel postgresql-devel glibc-devel curl-devel gcc automake postgresql
libidn-devel openssl-devel net-snmp-devel rpm-devel OpenIPMI-devel iksemel-devel libssh2-
devel openldap-devel
```

Пакет `iksemel-devel` используется для отправки Jabber-сообщений. Это действительно очень полезная возможность, так как позволяет серверу Zabbix посылать сообщения в чат. Кроме того, Jabber поддерживается в Zabbix как один из типов средств оповещения, для которых есть возможность настроить рабочие часы, чтобы избежать получения сообщений, когда вы находитесь не на работе.

Настройка сервера

Для нормальной работы сервера Zabbix требуется создать и настроить непривилегированную учетную запись. В любом случае, если демон запускается с привилегиями root, он автоматически переключается на работу с правами учетной записи Zabbix, если она имеется:

```
# groupadd zabbix
# useradd -m -s /bin/bash -g zabbix zabbix
# useradd -m -s /bin/bash -g zabbix zabbixsvr
```

Примечание. Никогда не запускайте сервер с привилегиями пользователя root, потому что это увеличивает риск плачевных последствий в случае взлома.

Предыдущие строки позволяют гарантировать повышенный уровень безопасности. Сервер и агент должны выполняться с разными учетными записями; в противном случае агент получит возможность доступа к конфигурации сервера Zabbix. Теперь, используя учетную запись Zabbix, можно загрузить и извлечь исходные тексты из файла `tar.gz`:

```
# wget http://sourceforge.net/projects/zabbix/files/ZABBIX%20Latest%20Stable/2.4.4/zabbix-
2.4.4.tar.gz/download -O zabbix-2.4.4.tar.gz

# tar -zxvf zabbix-2.4.4.tar.gz
```

Теперь займемся подготовкой исходных текстов к компиляции. Кстати, утилита настройки имеет справочный раздел:

```
# cd zabbix-2.4.3

# ./configure --help
```

В данном примере исходные тексты сервера мы настроим, указав следующие параметры:

```
# ./configure --enable-server --enable-agent --with-postgresql -with-libcurl --with-jabber
--with-net-snmp --enable-ipv6 --with-openipmi --with-ssh2 --with-ldap
```

Команды `zabbix_get` и `zabbix_send` генерируются, только если при настройке исходных текстов был указан флаг `--enable-agent`.

Если подготовка к компиляции завершилась без ошибок, на экране должны появиться примерно такие строки:

```
config.status: executing depfiles commands

Configuration:

  Detected OS:           linux-gnu
  Install path:          /usr/local
  Compilation arch:      linux

  Compiler:               gcc
  Compiler flags:        -g -O2 -I/usr/include -I/usr/include/rpm
-I/usr/local/include -I/usr/lib64/perl5/CORE -I. -I/usr/include -I/usr/
```

```
include -I/usr/include -I/usr/include

  Enable server:          yes
  Server details:
    With database:          PostgreSQL
    WEB Monitoring:         cURL
    Native Jabber:          yes
    SNMP:                   yes
    IPMI:                   yes
    SSH:                    yes
    ODBC:                   no
    Linker flags:           -rdynamic -L/usr/lib64 -L/usr/lib64
-L/usr/lib -L/usr/lib -L/usr/lib
    Libraries: -lm -ldl -lrt -lresolv -lpq -liksemel
-lnetsnmp -lssh2 -lOpenIPMI -lOpenIPMIposix -lldap -llber -lcurl

  Enable proxy:           no

  Enable agent:           yes

  Agent details:
    Linker flags:           -rdynamic -L/usr/lib
    Libraries:              -lm -ldl -lrt -lresolv -lldap -llber -lcurl

  Enable Java gateway:    no
  LDAP support:           yes
  IPv6 support:           yes

**********************************************************
*                  Now run 'make install'                *
*                                                        *
*                Thank you for using Zabbix!             *
*                  <http://www.zabbix.com>               *
**********************************************************
```

Мы пока запустим только команду `make`, без команды `make install`. Чтобы определить другой каталог для установки сервера Zabbix, используйте флаг `--prefix` команды `configure`, например: `--prefix=/opt/zabbix`. Теперь следуйте инструкциям в разделе «Установка и создание пакета».

Настройка агента

Чтобы подготовить исходные тексты агента к компиляции, выполните следующую команду:

```
# ./configure --enable-agent

# make
```

Если передать утилите `configure` ключ `--enable-static`, команда `make` выполнит статическую компоновку агента с библиотеками, благодаря чему выполняемому файлу для запуска не нужны будут внешние библиотеки; этот прием может пригодиться для подготовки агента, способного выполняться в разных версиях Linux.

Установка и создание пакета

В двух предыдущих разделах мы закончили компиляцию исходных текстов и теперь готовы выполнить установку; для чего достаточно запустить следующую команду:

```
# make install
```

Но я рекомендую не торопиться с установкой, а воспользоваться программой `checkinstall`. Она создаст пакет Zabbix и установит его.

Загрузить программу можно по адресу ftp://ftp.pbone.net/mirror/ftp5.gwdg.de/pub/opensuse/repositories/home:/ikoinoba/CentOS_CentOS-6/x86_64/checkinstall-1.6.2-3.el6.1.x86_64.rpm.

Обратите внимание, что `checkinstall` – лишь один из возможных способов создания пакета, пригодного к распространению.

Можно также воспользоваться предварительно собранной программой `checkinstall`. Текущей, на момент написания этих строк, была версия `checkinstall-1.6.2-20.4.i686.rpm` (в Red Hat/CentOS); пакет также требует установки `rpm-build`, которую можно выполнить командой (с привилегиями root):

```
# yum install rpm-build rpmdevtools
```

После этого требуется создать необходимые каталоги:

```
# mkdir -p ~/rpmbuild/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
```

Наличие пакета многое упрощает; его легче распространять и обновлять, плюс можно создавать пакеты для разных видов диспетчеров пакетов: `rpm`, `deb` и `tgz`.

Программа `checkinstall` способна создавать пакеты для Debian (ключ `-D`), Slackware (ключ `-S`) и Red Hat (ключ `-R`). Это особенно удобно для создания пакета с агентом Zabbix (статически скомпонованным) и установки его на серверы в окружении.

Теперь необходимо приобрести привилегии пользователя root или воспользоваться командой `sudo`, чтобы создать пакет:

```
# checkinstall --nodoc -R --install=no -y
```

Если процедура создания прошла без проблем, должно появиться примерно такое сообщение:

```
**********************************************************************
Done. The new package has been saved to
/root/rpmbuild/RPMS/x86_64/zabbix-2.4.4-1.x86_64.rpm
You can install it in your system anytime using:
     rpm -i zabbix-2.4.4-1.x86_64.rpm
**********************************************************************
```

После этого выполните (с привилегиями root) следующую команду, чтобы установить пакет:

```
# rpm -i zabbix-2.4.4-1.x86_64.rpm
```

Теперь система Zabbix установлена. Двоичные файлы сервера установлены в каталог <prefix>/sbin, утилиты – в каталог <prefix>/bin, а man-страницы (страницы справочного руководства) – в каталог <prefix>/share.

Установка из пакетов

Для полноты обзора всех возможных методов установки рассмотрим шаги, которые требуется выполнить, чтобы установить Zabbix из предварительно собранных rpm-пакетов.

Сначала необходимо подключить репозиторий:

```
# rpm -ivh http://repo.zabbix.com/zabbix/2.4/rhel/6/x86_64/zabbix-2.4.4-1.el6.x86_64.rpm
```

Эта команда создаст файл /etc/yum.repos.d/zabbix.repo с параметрами репозитория и подключит его.

Заглянув в репозиторий Zabbix, можно увидеть, что внутри «неподдерживаемого» дерева http://repo.zabbix.com/non-supported/rhel/6/x86_64/ доступны следующие пакеты: iksemel, fping, libssh2 и snmptt.

Теперь, чтобы установить сервер Zabbix и веб-интерфейс, достаточно выполнить команду:

```
# yum install zabbix-server-pgsql
```

А на веб-сервере (не забыв подключить репозиторий):

```
# yum install zabbix-web-pgsql
```

Чтобы установить агента, требуется выполнить только одну команду:

```
# yum install zabbix-agent
```

Если вы решите использовать пакеты RPM, имейте в виду, что конфигурационные файлы в этом случае сохраняются в каталог /etc/zabbix/. Но на протяжении всей книги будет предполагаться, что эти файлы находятся в каталоге /usr/local/etc/.

Кроме того, если в месте развертывания агента Zabbix имеется действующий локальный брандмауэр, вам потребуется настроить правила iptables, чтобы обеспечить беспрепятственное прохождение трафика через порт агента Zabbix:

```
# iptables -I INPUT 1 -p tcp --dport 10050 -j ACCEPT
# iptables-save
```

Настройка сервера

Для настройки сервера нам потребуется проверить и отредактировать единственный файл:

```
/usr/local/etc/zabbix_server.conf
```

Конфигурационные файлы находятся в каталоге:

```
/usr/local/etc
```

В файле /usr/local/etc/zabbix_server.conf нужно изменить имя пользователя, пароль и имя базы данных; обратите внимание, что настройка базы данных будет выполнена ниже в этой главе, а пока просто укажите планируемые имя пользователя/пароль/имя базы данных. Итак, действуя с привилегиями учетной записи zabbix, откройте файл для редактирования:

```
# vi /usr/local/etc/zabbix_server.conf
```

и измените следующие параметры:

```
DBHost=localhost

DBName=zabbix

DBUser=zabbix

DBPassword=<укажите-здесь-свой-пароль>
```

Теперь сервер Zabbix настроен и практически готов к запуску. Местоположение файла zabbix_server.conf определяется переменной времени компиляции sysconfdir. Не забудьте ограничить доступ к конфигурационному файлу, выполнив следующую команду:

```
chmod 600/usr/local/etc/zabbix_server.conf
```

Для внешних сценариев отводится каталог:

```
/usr/local/share/zabbix/externalscripts
```

Имя этого каталога определяется переменной времени компиляции datadir.

Для сценариев оповещений отводится каталог:

```
/usr/local/share/zabbix/alertscripts
```

Его так же можно настроить во время компиляции, определив значение переменной времени компиляции datadir.

Теперь перейдем к настройке агента. В конфигурационном файле агента нужно указать IP-адрес сервера Zabbix. После этого добавьте две службы в соответствующие уровни запуска, чтобы обеспечить их активизацию в требуемый момент времени.

Для выполнения этой задачи нужно установить сценарии запуска/остановки:

- /etc/init.d/zabbix-agent;
- /etc/init.d/zabbix-proxy;
- etc/init.d/zabbix-server.

Эти сценарии находятся в каталоге misc:

```
zabbix-2.4.4/misc/init.d
```

В их числе – сценарии запуска для разных дистрибутивов Linux. Но это дерево каталогов не поддерживается и не тестируется, и в нем могут находиться устаревшие сценарии, не подходящие для современных версий Linux, поэтому желательно просмотреть их и протестировать перед установкой.

После добавления сценариев запуска/остановки в каталог /etc/init.d нужно включить их в список служб:

```
# chkconfig --add zabbix-server
# chkconfig --add zabbix-agentd
```

Теперь осталось лишь сообщить системе, в какие уровни запуска они должны быть помещены; в этой книге мы будем использовать уровни 3 и 5:

```
# chkconfig --level 35 zabbix-server on
# chkconfig --level 35 zabbix-agentd on
```

Также, если на компьютере с сервером Zabbix имеется действующий брандмауэр, необходимо настроить iptables, чтобы обеспечить беспрепятственное прохождение трафика через порт сервера Zabbix, для чего нужно выполнить следующую команду (с привилегиями root):

```
# iptables -I INPUT 1 -p tcp --dport 10051 -j ACCEPT
# iptables-save
```

Прямо сейчас мы не можем запустить сервер, потому что еще не настроена база данных.

Установка базы данных

После установки сервера Zabbix можно заняться установкой и настройкой сервера баз данных. Все шаги, описываемые ниже, выполняются на выделенном сервере. Прежде всего нужно установить сервер PostgreSQL. Проще всего для этой цели использовать пакет, входящий в состав дистрибутива, но я рекомендую использовать последнюю стабильную версию 9.x.

В Red Hat (RHEL 6.4) все еще используется версия PostgreSQL 8.x. Эта же версия по наследству применяется в CentOS и ScientificLinux. Версия PosgreSQL 9.x имеет множество преимуществ; на момент написания этих строк последней стабильной и готовой к промышленной эксплуатации была версия 9.2.

Чтобы установить версию PostgreSQL 9.4, выполните следующие шаги:

1. Найдите файлы .repo:
   - **Red Hat**: /etc/yum/pluginconf.d/rhnplugin.conf [main];
   - **CentOS**: /etc/yum.repos.d/CentOS-Base.repo, [base] и [updates].
2. Добавьте следующую строку в конец разделов, обозначенных выше:

   ```
   exclude=postgresql*
   ```

3. Откройте в браузере страницу http://yum.postgresql.org и найдите соответствующую версию пакета RPM. Например, чтобы установить PostgreSQL 9.4 в RHEL 6, выберите пакет http://yum.postgresql.org/9.4/redhat/rhel-6-x86_64/pgdg-redhat94-9.4-1.noarch.rpm.
4. Подключите репозиторий командой: yum localinstall http://yum.postgresql.org/9.4/redhat/rhel-6-x86_64/pgdg-centos94-9.4-1.noarch.rpm.
5. Выведите список пакетов в репозитории postgresql командой:

   ```
   # yum list postgres*
   ```

6. Отыщите требуемый пакет и установите его командой:

```
# yum install postgresql94 postgresql94-server postgresql94-contrib
```

7. После установки пакета необходимо инициализировать базу данных:

```
# service postgresql-9.4 initdb
```

Инициализацию можно выполнить также командой:

```
# /etc/init.d/postgresql-9.4 initdb
```

8. Теперь займемся настройкой параметров в файле `/var/lib/pgsql/9.4/data/postgresql.conf`. Нужно определить адрес и номер порта для приема запросов:

```
listen_addresses = '*'
port = 5432
```

Также нужно добавить пару строк для настройки `zabbix_db`, сразу за следующими строками:

```
# TYPE DATABASE     USER     ADDRESS          METHOD

# "local" is for Unix domain socket connections only
local   all         all                      trust
in /var/lib/pgsql/9.4/data/pg_hba.conf
```

```
# настройки для Zabbix
local   zabbix_db   zabbix                   md5
host    zabbix_db   zabbix   <CIDR-address>   md5
```

Ключевое слово `local` соответствует всем соединениям, выполняющимся с помощью сокетов из домена Unix (Unix-domain sockets). За ним следуют имя базы данных (`zabbix_db`), имя пользователя (`zabbix`) и метод аутентификации (в данном случае `md5`).
Ключевое слово `host` соответствует всем соединениям, выполняющимся по протоколу TCP/IP (включая SSL). За ним следуют имя базы данных (`zabbix_db`), имя пользователя (`zabbix`), маска сети (определяет хосты, которым разрешено подключаться к базе данных) и метод аутентификации (в данном случае `md5`).

9. Маска сети, определяющая хосты, которым разрешено подключаться к базе данных, в данном случае должна быть обычной маской, потому что нам требуется открыть доступ к базе данных для веб-интерфейса (то есть веб-сервера) и сервера Zabbix, которые выполняются на других компьютерах. Примером такой маски может служить `10.6.0.0/24` (небольшая подсеть). Вероятнее всего, веб-интерфейс и сервер Zabbix будут находиться в другой сети, поэтому перечислите здесь все необходимые сети и маски.
10. В заключение запустите сервер PosgreSQL командой:

```
# service postgresql-9.4 start
```

или:

```
# /etc/init.d/postgresql-9.4 start
```

Чтобы создать базу данных, нужно зарегистрироваться с учетной записью пользователя `postgres` (или пользователя в вашем дистрибутиве, который наделен полномочиями управления базами данных PostgreSQL). Создайте пользователя в базе данных (это наш пользователь `zabbix`) и подключитесь под именем этого пользователя к серверу баз данных, чтобы импортировать схему данных.

Ниже следуют команды, необходимые для импортирования:

```
# su - postgres
```

После регистрации с учетной записью пользователя `postgres` можно создать базу данных (в данном случае базу данных с именем `zabbix_db`):

```
-bash-4.1$ psql

postgres=# CREATE USER zabbix WITH PASSWORD '<ПАРОЛЬ-ПОЛЬЗОВАТЕЛЯ-БАЗЫ-ДАННЫХ>';

CREATE ROLE

postgres=# CREATE DATABASE zabbix_db WITH OWNER zabbix ENCODING='UTF8';

CREATE DATABASE

postgres=# \q
```

Сценарии создания базы данных находятся в каталоге `/database/postgresql`, куда извлекались файлы с исходным кодом. Их нужно установить в точности в следующем порядке:

```
# cat schema.sql | psql -h <IP-АДРЕС-СЕРВЕРА-БАЗ-ДАННЫХ> -W -U zabbix zabbix_db
# cat images.sql | psql -h <IP-АДРЕС-СЕРВЕРА-БАЗ-ДАННЫХ> -W -U zabbix zabbix_db
# cat data.sql | psql -h <IP-АДРЕС-СЕРВЕРА-БАЗ-ДАННЫХ> -W -U zabbix zabbix_db
```

Параметр командной строки `-h <IP-АДРЕС-СЕРВЕРА-БАЗ-ДАННЫХ>` команды `psql` помогает обойти параметр `local` в стандартном конфигурационном файле `/var/lib/pgsql/9.4/data/pg_hba.conf`.

Теперь можно запустить сервер Zabbix и проверить связку сервер/база данных:

```
# /etc/init.d/zabbix-server start

Starting Zabbix server:                                [ OK ]
```

Заглянув в файл журнала, можно получить более полную информацию о происходящем на сервере. Там должны находиться следующие строки (по умолчанию файл журнала размещается в `/tmp/zabbix_server.log`):

```
26284:20150114:034537.722 Starting Zabbix Server. Zabbix 2.4.4 (revision 51175).
26284:20150114:034537.722 ****** Enabled features ******
26284:20150114:034537.722 SNMP monitoring: YES
26284:20150114:034537.722 IPMI monitoring: YES
26284:20150114:034537.722 WEB monitoring: YES
26284:20150114:034537.722 VMware monitoring: YES
26284:20150114:034537.722 Jabber notifications: YES
26284:20150114:034537.722 Ez Texting notifications: YES
```

```
26284:20150114:034537.722 ODBC: YES
26284:20150114:034537.722 SSH2 support: YES
26284:20150114:034537.722 IPv6 support: YES
26284:20150114:034537.725 ******************************
26284:20150114:034537.725 using configuration file: /usr/local/etc/zabbix/zabbix_server.conf
26284:20150114:034537.745 current database version (mandatory/optional): 02040000/02040000
26284:20150114:034537.745 required mandatory version: 02040000
26284:20150114:034537.763 server #0 started [main process]
26289:20150114:034537.763 server #1 started [configuration syncer #1]
26290:20150114:034537.764 server #2 started [db watchdog #1]
26291:20150114:034537.764 server #3 started [poller #1]
26293:20150114:034537.765 server #4 started [poller #2]
26294:20150114:034537.766 server #5 started [poller #3]
26296:20150114:034537.770 server #7 started [poller #5]
26295:20150114:034537.773 server #6 started [poller #4]
26297:20150114:034537.773 server #8 started [unreachable poller #1]
26298:20150114:034537.779 server #9 started [trapper #1]
26300:20150114:034537.782 server #11 started [trapper #3]
26302:20150114:034537.784 server #13 started [trapper #5]
26301:20150114:034537.786 server #12 started [trapper #4]
26299:20150114:034537.786 server #10 started [trapper #2]
26303:20150114:034537.794 server #14 started [icmp pinger #1]
26305:20150114:034537.790 server #15 started [alerter #1]
26312:20150114:034537.822 server #18 started [http poller #1]
26311:20150114:034537.811 server #17 started [timer #1]
26310:20150114:034537.812 server #16 started [housekeeper #1]
26315:20150114:034537.829 server #20 started [history syncer #1]
26316:20150114:034537.844 server #21 started [history syncer #2]
26319:20150114:034537.847 server #22 started [history syncer #3]
26321:20150114:034537.852 server #24 started [escalator #1]
26320:20150114:034537.849 server #23 started [history syncer #4]
26326:20150114:034537.868 server #26 started [self-monitoring #1]
26325:20150114:034537.866 server #25 started [proxy poller #1]
26314:20150114:034537.997 server #19 started [discoverer #1]
```

Вообще говоря, место хранения по умолчанию файла журнала выбрано не самое лучшее, потому что каталог `/tmp` автоматически очищается при каждой перезагрузке, а сами файлы не архивируются.

Место по умолчанию можно изменить в конфигурационном файле `/etc/zabbix_server.conf`. Достаточно просто изменить параметр:

```
### Option: LogFile
LogFile=/var/log/zabbix/zabbix_server.log
```

Не забудьте после этого создать структуру каталогов, выполнив следующую команду с привилегиями root:

```
# mkdir -p /var/log/zabbix
# chown zabbixsvr:zabbixsvr /var/log/zabbix
```

Еще один важный аспект – настройка автоматической ротации журнала в logrotate. Это легко реализуется простым добавлением конфигурационного файла в каталог /etc/logrotate.d/.

Создайте файл, выполнив следующую команду с привилегиями root:

```
# vim /etc/logrotate.d/zabbix-server
```

и добавьте в него следующие строки:

```
/var/log/zabbix/zabbix_server.log {
        missingok
        monthly
        notifempty
        compress
        create 0664 zabbix zabbix
}
```

После этого перезапустите сервер Zabbix следующей командой (с привилегиями root):

```
# /etc/init.d/zabbix-server restart
Shutting down Zabbix server:                    [ OK ]
Starting Zabbix server:                         [ OK ]
```

Кроме того, проверьте, запущен ли сервер с привилегиями соответствующего пользователя:

```
# ps aux | grep "[z]abbix_server"
502 28742 1  0 13:39 ?    00:00:00 /usr/local/sbin/zabbix_server
502 28744 28742 0 13:39 ? 00:00:00 /usr/local/sbin/zabbix_server
502 28745 28742 0 13:39 ? 00:00:00 /usr/local/sbin/zabbix_server
...
```

Вывод команды показывает, что zabbix_server выполняется с привилегиями пользователя 502. Сделаем еще шаг и проверим, соответствует ли идентификатор 502 пользователю, созданному нами выше:

```
# getent passwd 502
zabbixsvr:x:502:501::/home/zabbixsvr:/bin/bash
```

Как следует из полученной строки, все замечательно.

Иногда в журнале можно увидеть строку с ошибкой:

```
28487:20130609:133341.529 Database is down. Reconnecting in 10 seconds.
```

Эта проблема может быть вызвана несколькими причинами:

- брандмауэр (локальный, на нашем сервере, или находящийся в сетевой инфраструктуре) не пропускает сетевых пакетов;
- допущены ошибки в настройках сервера PostgreSQL;
- допущены ошибки в файле zabbix_server.conf.

Можно попробовать локализовать проблему, выполнив следующую команду на сервере баз данных:

```
psql -h <IP-АДРЕС-СЕРВЕРА-БАЗ-ДАННЫХ> -U zabbix zabbix_db
Password for user zabbix:
psql (9.4) Type "help" for help
```

Если соединение установилось, попробуйте выполнить ту же команду на сервере Zabbix; если попытка окончилась неудачей, проверьте настройки брандмауэра. Если команда вывела сообщение об ошибке аутентификации, проверьте настройки в файле `pg_hba.conf`.

Следующий шаг – проверка настроек локального брандмауэра и `iptables`. Проверьте, открыт ли порт PostgreSQL на сервере баз данных. Если порт не открыт, добавьте правило, выполнив следующую команду с привилегиями root:

```
# iptables -I INPUT 1 -p tcp --dport 5432 -j ACCEPT
# iptables-save
```

Теперь проверим, как выполняется запуск/остановка Zabbix. Сценарии, что приводятся ниже, предполагают, что для управления сервером и агентами созданы разные пользователи.

Следующий сценарий запуска прекрасно работает с версией, скомпилированной без флага `--prefix`, и при наличии в системе учетной записи пользователя `zabbixsvr`. Если вы используете иные настройки, измените путь к файлу сервера и имя пользователя:

```
exec=/usr/local/sbin/zabbix_server
zabbixsrv=zabbixsvr
```

Создайте в каталоге `/etc/init.d` файл `zabbix-server` и добавьте в него следующие строки:

```
#!/bin/sh
#
# chkconfig: - 85 15
# description: Zabbix server daemon
# config: /usr/local/etc/zabbix_server.conf
#

### BEGIN INIT INFO
# Provides: zabbix
# Required-Start: $local_fs $network
# Required-Stop: $local_fs $network
# Default-Start:
# Default-Stop: 0 1 2 3 4 5 6
# Short-Description: Start and stop Zabbix server
# Description: Zabbix server
### END INIT INFO

# Подключить библиотеку функций.
```

```
. /etc/rc.d/init.d/functions

exec=/usr/local/sbin/zabbix_server
prog=${exec##*/}
lockfile=/var/lock/subsys/zabbix
syscf=zabbix-server
```

Следующий параметр, zabbixsvr, используемый в функции start(), определяет пользователя, с привилегиями которого должен быть запущен сервер Zabbix:

```
zabbixsrv=zabbixsvr
[ -e /etc/sysconfig/$syscf ] && . /etc/sysconfig/$syscf

start()
{
    echo -n $"Starting Zabbix server: "
```

В предыдущем фрагменте настраивается пользователь (владелец процесса сервера Zabbix), который затем используется в функции start():

```
    daemon --user $zabbixsrv $exec
```

Не забудьте изменить принадлежность конфигурационного файла и файла журнала Zabbix. Эта мера предотвратит доступ к конфиденциальным данным со стороны обычных пользователей. Путь к файлу журнала определяется параметром Logfile в конфигурационном файле /usr/local/etc/zabbix_server.conf.

```
    rv=$?
    echo
    [ $rv -eq 0 ] && touch $lockfile
    return $rv
}

stop()
{
    echo -n $"Shutting down Zabbix server: "
```

Внутри функции stop не требуется определять пользователя, так как сценарий запуска/остановки выполняется с привилегиями root. Поэтому можно просто использовать команду killproc $prog:

```
    killproc $prog
    rv=$?
    echo
    [ $rv -eq 0 ] && rm -f $lockfile
    return $rv
}

restart()
{
    stop
    start
}
```

```
case "$1" in
    start|stop|restart)
        $1
        ;;
    force-reload)
        restart
        ;;
    status)
        status $prog
        ;;
    try-restart|condrestart)
        if status $prog >/dev/null ; then
            restart
        fi
        ;;
    reload)
        action $"Service ${0##*/} does not support the reload action: " /bin/false
        exit 3
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|try-restart|forcereload}"
        exit 2
        ;;
    esac
```

Следующий сценарий запуска прекрасно работает с версией, скомпилированной без флага `--prefix`, и при наличии в системе учетной записи пользователя `zabbix`. Если вы используете иные настройки, измените путь к файлу сервера и имя пользователя:

```
exec=/usr/local/sbin/zabbix_agentd
zabbix_usr=zabbixsvr
```

Создайте в каталоге `/etc/init.d` файл `zabbix-agent` и добавьте в него следующие строки:

```
#!/bin/sh
#
# chkconfig: - 86 14
# description: Zabbix agent daemon
# processname: zabbix_agentd
# config: /usr/local/etc/zabbix_agentd.conf
#

### BEGIN INIT INFO
# Provides: zabbix-agent
# Required-Start: $local_fs $network
# Required-Stop: $local_fs $network
# Should-Start: zabbix zabbix-proxy
# Should-Stop: zabbix zabbix-proxy
```

```
# Default-Start:
# Default-Stop: 0 1 2 3 4 5 6
# Short-Description: Start and stop Zabbix agent
# Description: Zabbix agent
### END INIT INFO

# Подключить библиотеку функций.
. /etc/rc.d/init.d/functions

exec=/usr/local/sbin/zabbix_agentd
prog=${exec##*/}
syscf=zabbix-agent
lockfile=/var/lock/subsys/zabbix-agent
```

Следующий параметр, `zabbix_usr`, определяет пользователя, с привилегиями которого должен быть запущен агент Zabbix:

```
zabbix_usr=zabbix
[ -e /etc/sysconfig/$syscf ] && . /etc/sysconfig/$syscf

start()
{
    echo -n $"Starting Zabbix agent: "
```

Следующая команда использует значение переменной `zabbix_usr`, что позволяет нам иметь две разные учетные записи – одну для сервера и одну для агента – и закрыть доступ к настройкам в файле `zabbix_server.conf`, который содержит пароль доступа к базе данных, для агента Zabbix:

```
    daemon --user $zabbix_usr $exec
    rv=$?
    echo
    [ $rv -eq 0 ] && touch $lockfile
    return $rv
}

stop()
{
    echo -n $"Shutting down Zabbix agent: "
    killproc $prog
    rv=$?
    echo
    [ $rv -eq 0 ] && rm -f $lockfile
    return $rv
}

restart()
{
    stop
    start
}

case "$1" in
```

```
start|stop|restart)
    $1
    ;;
force-reload)
    restart
    ;;
status)
    status $prog
    ;;
try-restart|condrestart)
    if status $prog >/dev/null ; then
        restart
    fi
    ;;
reload)
    action $"Service ${0##*/} does not support the reload action: " /bin/false
    exit 3
    ;;
*)
    echo $"Usage: $0 {start|stop|status|restart|try-restart|forcereload}"
    exit 2
    ;;
esac
```

Теперь у нас имеются агент, выполняющийся с привилегиями `zabbix`, и сервер, выполняющийся с привилегиями `zabbixsvr`:

```
zabbix_usr_ 4653 1    0 15:42 ? 00:00:00 /usr/local/sbin/zabbix_agentd
zabbix_usr  4655 4653 0 15:42 ? 00:00:00 /usr/local/sbin/zabbix_agentd
zabbixsvr   4443 1    0 15:32 ? 00:00:00 /usr/local/sbin/zabbix_server
zabbixsvr   4445 4443 0 15:32 ? 00:00:00 /usr/local/sbin/zabbix_server
```

Подготовка базы данных

Zabbix использует интересный способ сохранения постоянного размера базы данных. В действительности размер базы данных определяется:

- количеством значений, обрабатываемых каждую секунду;
- параметрами настройки.

Zabbix использует два пути хранения данных:

- историю (в хронологической последовательности);
- динамику (тренд).

История хранит все собранные данные (независимо от их типов); динамика (тренд) – только числовые данные: минимальное, максимальное и среднее значения за час (чтобы сделать процедуру сохранения максимально легковесной).

Строковые элементы – символы, сообщения, текст – не могут сохраняться в тренде, потому что они могут хранить только числовые значения.

В Zabbix имеется процесс `housekeeper`, который отвечает за поддержание постоянного размера базы данных. Я настоятельно рекомендую хранить историю за как

можно более короткий период, чтобы не переполнять базу данных большим объемом информации, а для длительного хранения сведений использовать тренды.

Теперь, учитывая, что Zabbix также используется для планирования наращивания мощностей, нам нужно рассмотреть основные критерии, определяющие размер базы данных, хотя бы на период одного бизнес-цикла. Обычно минимальная продолжительность бизнес-цикла составляет один год, но я настоятельно рекомендую хранить тренд не менее 2 лет. Эти тренды можно использовать при открытии и закрытии бизнеса для количественной оценки расходов за определенный период.

Если динамика изменений (тренд) установлена в `0`, сервер не будет вычислять и сохранять тренд. Если история установлена в `0`, сервер будет считать триггеры только для последнего значения, потому что исторические значения не сохраняются в базе данных.

Нередко при объединении данных возникает проблема, связанная с положительными или отрицательными пиками в почасовых трендах, которые могут приводить к получению неверного среднего значения за час.

В Zabbix используется интеллектуальная реализация трендов. Начнем знакомство с ней со сценария, создающего таблицы для хранения трендов:

```
CREATE TABLE trends(
itemid bigin NOT NULL, clock integer DEFAULT '0'
NOT NULL, num integer DEFAULT '0'
NOT NULL, value_min numeric(16, 4) DEFAULT '0.0000'
NOT NULL, value_avg numeric(16, 4) DEFAULT '0.0000'
NOT NULL, value_max numeric(16, 4) DEFAULT '0.0000'
NOT NULL, PRIMARY KEY(itemid, clock));

CREATE TABLE trends_uint(
Itemid bigint NOT NULL, Clock integer DEFAULT '0'
NOT NULL, Num integer DEFAULT '0'
NOT NULL, value_min numeric(20) DEFAULT '0'
NOT NULL, value_avg numeric(20) DEFAULT '0'
NOT NULL, value_max numeric(20) DEFAULT '0'
NOT NULL, PRIMARY KEY(itemid, clock));
```

Как видите, для хранения трендов в базе данных Zabbix используются две таблицы:

- `trends`;
- `trends_uint`.

Первая таблица, `trends`, используется для хранения вещественных значений, а вторая, `trends_uint`, – целочисленных без знака. Обе таблицы следуют идее хранения следующих значений за каждый час:

- минимальное (`value_min`);
- максимальное (`value_max`);
- среднее (`value_avg`).

Такая организация помогает находить и отображать тренды в графическом виде и определять влияние пиков на среднее значение и величину этого влияния. Для хранения исторических значений используются следующие таблицы:

- `history`: хранит числовые (вещественные) данные;
- `history_log`: хранит журнал (например, текстовое поле неограниченной длины);
- `history_str`: хранит строки (длиной до 255 символов);
- `history_text`: хранит текстовые значения (также текстовое поле неограниченной длины);
- `history_uint`: хранит числовые (целочисленные без знака) значения.

Оценка размера базы данных

Оценка окончательного размера базы данных – не самая простая задача, потому что заранее трудно предсказать количество элементов данных и частоту обновлений в секунду, которые потребуются для мониторинга инфраструктуры, и сколько событий будет генерироваться в ней. Чтобы упростить задачу, возьмем за основу самый тяжелый случай, когда каждую секунду генерируется одно событие.

В общем случае размер базы данных определяется:

- **элементами данных**: точнее, их количеством;
- **частотой изменений**: средней частотой изменений элементов данных;
- **пространством, занимаемым хранимыми значениями**: эта величина зависит от СУБД.

Одни и те же данные в разных базах данных могут занимать разное пространство, но мы можем упростить задачу, взяв за основу максимальный объем. Так, будем полагать, что для хранения одного исторического значения используется 50 байт, для хранения одного значения в таблице `trend` используется 128 байт, а для хранения одного события – 130 байт.

Общий объем занимаемого пространства вычисляется по формуле:

Конфигурация + История + Тренды + События.

Теперь рассмотрим каждый компонент формулы в отдельности:

- **Конфигурация**: под этим элементом понимается конфигурация сервера Zabbix, веб-интерфейса и всех конфигурационных параметров, хранимых в базе данных; обычно этот элемент занимает порядка 10 МБ;
- **История**: объем истории вычисляется с применением следующей формулы:

```
Срок_хранения_истории_в_днях  * (элементов/период_изменений) * 24 * 3600 * 50 байт
(для хранения одного исторического значения)
```

- **Тренды**: объем трендов вычисляется с применением следующей формулы:

```
дней * (элементов/3600) * 24 * 3600 * 128 байт (для хранения одного значения тренда)
```

- **События**: объем событий вычисляется с применением следующей формулы:

```
дней * событий * 24 * 3600 * 130 байт (для хранения одного события)
```

Теперь вернемся к нашему практическому примеру. Если предположить, что у нас имеется 5000 элементов данных, обновляемых с частотой один раз в минуту, и требуется хранить историю за последние 30 дней, тогда потребуется:

```
30 * 5000/60 * 24 * 3600 * 50 = 10.8 ГБ
```

50 – это объем в байтах, занимаемый одним историческим значением.

Как отмечалось выше, для простоты мы взяли самый худший сценарий (одно событие в секунду). Давайте определим, какой объем потребуется для хранения событий в течение 5 лет. Подставив значения в формулу:

```
дней * событий * 24 * 3600 * 130 байт (для хранения одного события)
```

получаем:

```
5 * 365 * 24 * 3600 * 130 = 15.7 ГБ
```

130 – это объем в байтах, занимаемый одним событием.

Для случая хранения трендов в течение одного года подставим значения в формулу:

```
дней * (элементов/3600) * 24 * 3600 * 128 байт (для хранения одного значения тренда)
```

получим:

```
365 * 5000/3600 * 24 * 3600 * 128 = 5.3 ГБ
```

или **26,7 ГБ** для срока в 5 лет.

128 – это объем в байтах, занимаемый одним значением в трендах.

В табл. 1.1 приводится соотношение между числом дней хранения информации и занимаемым ею объемом:

Таблица 1.1 ❖ Соотношение между числом дней хранения информации и занимаемым ею объемом

Компонент	Срок хранения в днях	Занимаемый объем
История	30	10,8 ГБ
События	1825 (5 лет)	15,7 ГБ
Тренды	1825 (5 лет)	26,7 ГБ
Всего	–	53,2 ГБ

Надо понимать, что вычисленный объем – это не начальный размер базы данных, а конечный, который будет достигнут через 5 лет. Кроме того, мы предположили, что история будет храниться 30 дней, то есть это значение можно уменьшить, учитывая, что наши тренды хранят информацию за каждый час.

Политика продолжительности хранения истории и трендов может гибко изменяться для каждого элемента. Это означает, что можно создать шаблон с элементами, имеющими разную длительность хранения исторических данных. Обычно история хранится 7 дней, но мы можем изменить это значение и хранить историю дольше недели.

В нашем примере мы рассмотрели худший сценарий, требующий хранения истории 30 дней, но вообще в больших окружениях рекомендуется хранить историю 7 дней или даже меньше. Приняв за основу, что элемент данных изменяется каждые 60 секунд и история должна храниться 7 дней, получаем *(период изменения) * (часов в дне) * (дней в истории) = 60 * 24 * 7 = 10 080.*

То есть в течение недели для каждого элемента данных будет создано 10 080 записей, откуда несложно вывести общий размер базы данных.

На рис. 1.3 показана форма с параметрами настройки элемента данных.

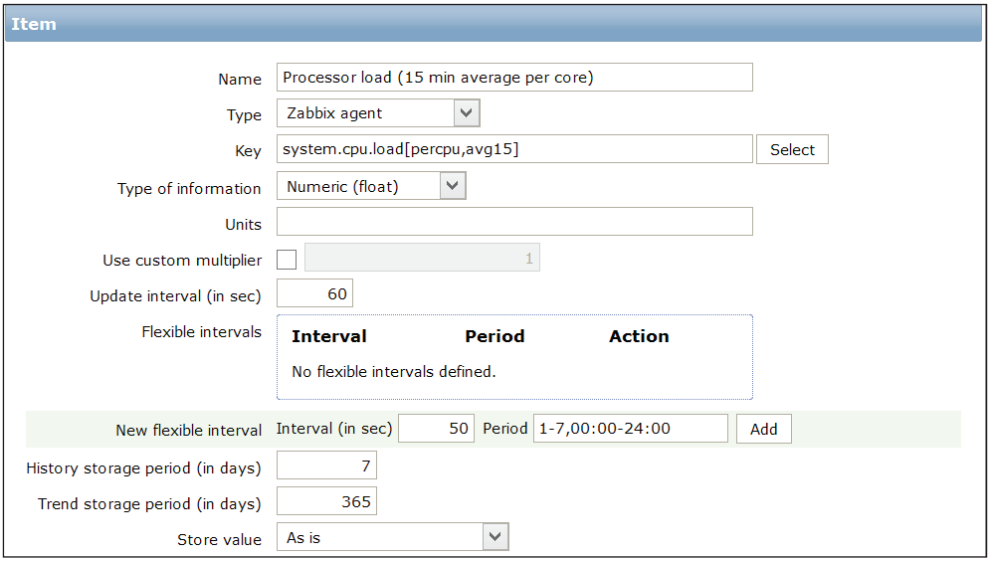


Рис. 1.3 ❖ Форма настройки элемента данных

Очистка истории

Очистка истории иногда превращается в довольно сложный процесс. С ростом базы данных требуется все больше и больше времени на выполнение этой операции. Эту проблему можно решить с помощью функции `delete_history()` в базе данных.

Существует возможность существенно увеличить скорость обслуживания и предотвратить падение производительности операций с самыми тяжелыми таблицами: `history`, `history_uint`, `trends` и `trends_uint`.

Решение заключается в использовании механизма PostgreSQL секционирования (partitioning) таблиц и их деления на помесячной основе. На рис. 1.4 схематически изображена стандартная, несекционированная таблица `history` в базе данных.

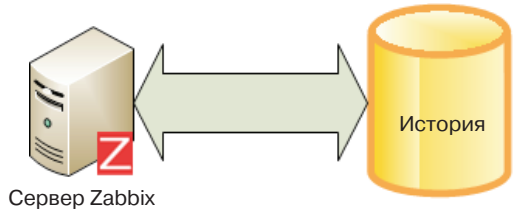


Рис. 1.4 ❖ Стандартная, несекционированная таблица `history`

На рис. 1.5. показано, как в базе данных хранится секционированная таблица.

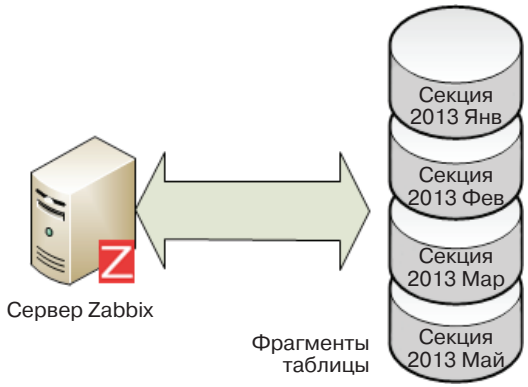


Рис. 1.5 ❖ Секционированная таблица `history`

Суть секционирования заключается в делении большой логической таблицы на множество более мелких физических фрагментов, или разделов. Это дает несколько преимуществ.

- Производительность запросов может существенно увеличиваться, когда все извлекаемые данные находятся в пределах одного раздела (секции).
- Секционирование уменьшает размер индексов и тем самым увеличивает вероятность, что интенсивно используемые разделы уместятся в оперативной памяти.
- Удаляя целые разделы, можно выполнять массовое удаление и немедленно освобождать пространство без его фрагментации, не вызывая тяжелую перестройку индексов. Команда `delete partition` также не вызывает огромных накладных расходов, так свойственных командам `vacuum` и `delete`.
- Когда выполняются обновления или требуется доступ к большей части раздела, последовательное сканирование часто оказывается эффективнее использования индексов с произвольным доступом или чтения индексов вразброс.

Все эти преимущества проявляются, только когда таблица имеет действительно большой размер. Сильной стороной такой архитектуры является возможность

прямого доступа СУБД к требуемому разделу, и операция удаления заключается в простом удалении раздела. Удаление раздела выполняется быстро и не требует большого количества ресурсов.

К сожалению, система Zabbix не способна управлять разделами, поэтому необходимо отключить встроенную функцию очистки истории и использовать для этого внешние процессы.

Прием секционирования, описанный здесь, имеет определенные преимущества перед другими решениями:

- не требует подготовки базы данных к секционированию для нужд Zabbix;
- не требует добавлять задания для планировщика cron, которые создавали бы таблицы с опережением;
- прост в реализации.

Данный метод подготавливает разделы в соответствии с требуемой схемой секционирования и следующими соглашениями:

- ежедневные разделы имеют форму `partitions.tablename_pYYYYMMDD`;
- ежемесячные разделы имеют форму `partitions.tablename_pYYYYMM`.

Все упоминаемые здесь сценарии доступны по адресу: https://github.com/smartmarmot/Mastering_Zabbix.

Итак, прежде всего нужно создать схему для размещения всех секционированных таблиц. Затем, в разделе `psql`, требуется выполнить команду:

```
CREATE SCHEMA partitions AUTHORIZATION zabbix;
```

Нам также понадобится функция создания раздела. Для этого, подключившись к Zabbix, выполните следующий код:

```
CREATE OR REPLACE FUNCTION trg_partition()
RETURNS TRIGGER AS
$BODY$
DECLARE
    prefix text:= 'partitions.';
    timeformat text;
    selector text;
    _interval INTERVAL;
    tablename text;
    startdate text;
    enddate text;
    create_table_part text;
    create_index_part text;
BEGIN
selector = TG_ARGV[0];
IF selector = 'day'
    THEN
    timeformat:= 'YYYY_MM_DD';
ELSIF selector = 'month'
    THEN
    timeformat:= 'YYYY_MM';
```

```
END IF;

_interval:= '1 ' || selector;
tablename:= TG_TABLE_NAME || '_p' ||
            TO_CHAR(TO_TIMESTAMP(NEW.clock),
                    timeformat);

EXECUTE 'INSERT INTO ' || prefix || quote_ident(tablename) || ' SELECT
($1).*'
USING NEW;
RETURN NULL;

EXCEPTION
WHEN undefined_table THEN
startdate:= EXTRACT(epoch FROM
    date_trunc(selector, TO_TIMESTAMP(NEW.clock)));
enddate:= EXTRACT(epoch FROM
    date_trunc(selector, TO_TIMESTAMP(NEW.clock) + _interval));

create_table_part:= 'CREATE TABLE IF NOT EXISTS ' ||
    prefix || quote_ident(tablename) ||
    ' (CHECK ((clock >= ' || quote_literal(startdate) ||
    ' AND clock < ' || quote_literal(enddate) ||
    '))) INHERITS (' || TG_TABLE_NAME || ')';

create_index_part:= 'CREATE INDEX ' || quote_ident(tablename) ||
    '_1 on ' || prefix || quote_ident(tablename) ||
    '(itemid,clock)';

EXECUTE create_table_part;
EXECUTE create_index_part;

--вставить снова
EXECUTE 'INSERT INTO ' || prefix || quote_ident(tablename) ||
        ' SELECT ($1).*'
USING NEW;
RETURN NULL;

END;

$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
ALTER FUNCTION trg_partition()
    OWNER TO zabbix;
```

В базе данных должна быть создана учетная запись для пользователя `zabbix`. Если вы используете другую роль/учетную запись, измените последнюю строку в листинге выше:

```
ALTER FUNCTION trg_partition()
OWNER TO <подставьте сюда имя пользователя, владельца базы данных>;
```

Теперь необходимо подключить триггер к каждой секционируемой таблице. Этот триггер выполняет инструкцию INSERT, и, если раздел еще не готов или не создан, функция создаст его перед выполнением инструкции INSERT:

```
CREATE TRIGGER partition_trg BEFORE INSERT ON history
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('day');
CREATE TRIGGER partition_trg BEFORE INSERT ON history_sync
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('day');
CREATE TRIGGER partition_trg BEFORE INSERT ON history_uint
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('day');
CREATE TRIGGER partition_trg BEFORE INSERT ON history_str_sync
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('day');
CREATE TRIGGER partition_trg BEFORE INSERT ON history_log
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('day');
CREATE TRIGGER partition_trg BEFORE INSERT ON trends
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('month');
CREATE TRIGGER partition_trg BEFORE INSERT ON trends_uint
    FOR EACH ROW EXECUTE PROCEDURE trg_partition('month');
```

В настоящий момент нам осталось только создать свою функцию очистки истории и отключить встроенную в Zabbix:

```
CREATE OR REPLACE FUNCTION delete_partitions(
    intervaltodelete INTERVAL, tabletype text)
RETURNS text AS
$BODY$
DECLARE
result RECORD ;
prefix text := 'partitions.';
table_timestamp TIMESTAMP;
delete_before_date DATE;
tablename text;

BEGIN
    FOR result IN SELECT * FROM pg_tables
                  WHERE schemaname = 'partitions' LOOP
        table_timestamp := TO_TIMESTAMP(substring(
            result.tablename FROM '[0-9_]*$'), 'YYYY_MM_DD');
        delete_before_date := date_trunc('day',
            NOW() - intervalToDelete);
        tablename := result.tablename;
        IF tabletype != 'month' AND tabletype != 'day' THEN
            RAISE EXCEPTION
                'Please specify "month" or "day" instead of %',
                tabletype;
        END IF;

        --Убедиться, что имя таблицы имеет
        --формат (YYYY_MM_DD) или (YYYY_MM)
        IF LENGTH(substring(result.tablename FROM '[0-9_]*$')) = 10
```

```
           AND tabletype = 'month' THEN
            --Это дневной раздел YYYY_MM_DD
            -- RAISE NOTICE 'Skipping table % when trying to delete "%" partitions (%)',
result.tablename, tabletype, length(substring(result.tablename from '[0-9_]*$'));
            CONTINUE;
        ELSIF LENGTH(substring(result.tablename FROM '[0-9_]*$')) = 7
              AND tabletype = 'day' THEN
            --Это месячный раздел
            --RAISE NOTICE 'Skipping table % when trying to delete "%" partitions (%)',
result.tablename, tabletype, length(substring(result.tablename from '[0-9_]*$'));
            CONTINUE;
        ELSE
            --Таблица имеет верный тип.
            --Проверить необходимость ее удаления
            --RAISE NOTICE 'Checking table %', result.tablename;
        END IF;

        IF table_timestamp <= delete_before_date THEN
            RAISE NOTICE 'Deleting table %', quote_ident(tablename);
            EXECUTE 'DROP TABLE ' || prefix ||
                    quote_ident(tablename) || ';';
        END IF;
    END LOOP;
RETURN 'OK';
END;

$BODY$
    LANGUAGE plpgsql VOLATILE
    COST 100;
ALTER FUNCTION delete_partitions(INTERVAL, text)
    OWNER TO zabbix;
```

Теперь, когда необходимые функции очистки определены, их выполнение нужно запланировать с помощью `crontab`:

```
@daily psql -h<your database host here> -d zabbix_db -q -U zabbix -c "SELECT delete_
partitions('7 days', 'day')"
@daily psql -h<your database host here> -d zabbix_db -q -U zabbix -c "SELECT delete_
partitions('24 months', 'month')"
```

Эти задания должны определяться в планировщике `crontab` на сервере баз данных. В данном примере предполагается хранить историю 7 дней, а тренды – 24 месяца.

Теперь отключим функцию обслуживания, встроенную в Zabbix. В Zabbix 2.4 для этого лучше всего использовать веб-интерфейс, где следует выбрать раздел **Administration ⇨ General ⇨ Housekeeper** (Администрирование ⇨ Общие ⇨ Очистка истории) и в открывшихся настройках отключить очистку для таблиц **Trends** и **History**, как показано на рис. 1.6.

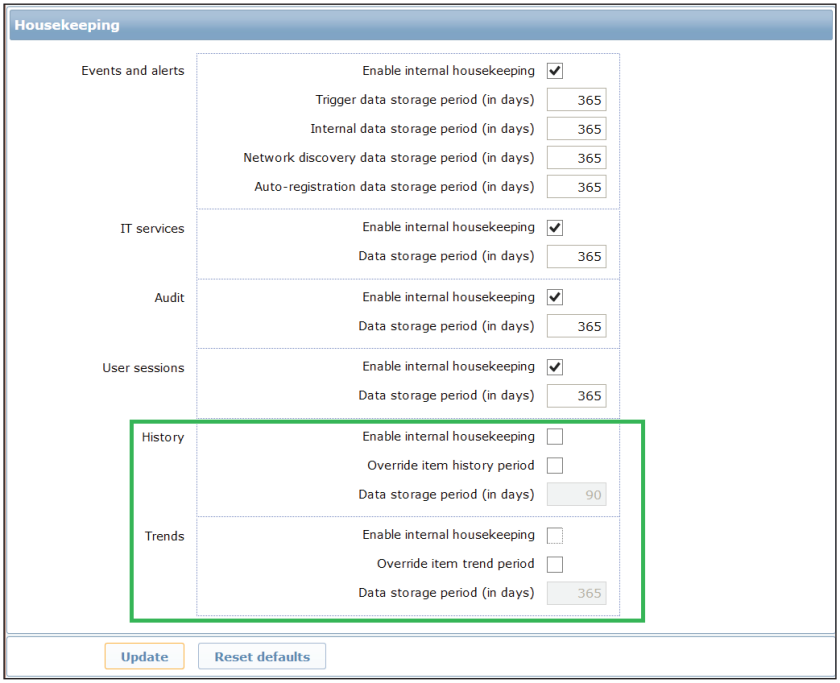


Рис. 1.6 ❖ Раздел с настройками очистки истории

Теперь встроенный механизм очистки не будет вызываться, а производительность должна возрасти. Чтобы сохранить базу данных максимально легковесной, можно предусмотреть очистку следующих таблиц:

- acknowledges;
- alerts;
- auditlog;
- events;
- service_alarms;

После выбора периода хранения данных необходимо добавить политику хранения; например, в данном примере данные будут храниться 2 года. Добавив следующие задания в crontab, можно обеспечить удаление записей, созданных более 63 072 000 секунд (2 лет) тому назад:

```
@daily psql -q -U zabbix -c "delete from acknowledges where clock < (SELECT (EXTRACT( epoch
FROM now() ) - 63072000))"

@daily psql -q -U zabbix -c "delete from alerts where clock < (SELECT (EXTRACT( epoch FROM
now() ) - 63072000))"

@daily psql -q -U zabbix -c "delete from auditlog where clock < (SELECT (EXTRACT( epoch
FROM now() ) - 62208000))"
```

```
@daily psql -q -U zabbix -c "delete from events where clock < (SELECT (EXTRACT( epoch FROM
now() ) - 62208000))"

@daily psql -q -U zabbix -c "delete from service_alarms where clock < (SELECT (EXTRACT(
epoch FROM now() ) - 62208000))"
```

Чтобы отключить очистку истории, нужно удалить триггеры:

```
DROP TRIGGER partition_trg ON history;
DROP TRIGGER partition_trg ON history_sync;
DROP TRIGGER partition_trg ON history_uint;
DROP TRIGGER partition_trg ON history_str_sync;
DROP TRIGGER partition_trg ON history_log;
DROP TRIGGER partition_trg ON trends;
DROP TRIGGER partition_trg ON trends_uint;
```

Обязательно проверьте и протестируйте эти изменения, чтобы убедиться, что они подходят для вашего случая. Кроме того, перед тестированием создайте резервную копию базы данных на всякий случай.

Веб-интерфейс

Установка веб-интерфейса выполняется очень просто; достаточно выполнить несколько элементарных шагов. Веб-интерфейс полностью написан на языке PHP, поэтому для его работы необходим веб-сервер, поддерживающий PHP; в данном случае мы будем использовать Apache с включенной поддержкой PHP.

Весь веб-интерфейс хранится в каталоге `frontends/php/`, который нужно скопировать в каталог `htdocs`:

```
/var/www/htdocs
```

Копирование содержимого каталога можно выполнить следующими командами:

```
# mkdir <htdocs>/zabbix
# cd frontends/php
# cp -a . <htdocs>/zabbix
```

Для выполнения этих команд могут потребоваться соответствующие привилегии, так как владельцем всех этих файлов является веб-сервер Apache и имеется зависимость от настроек в файле конфигурации `httpd`.

Мастер настройки – настройка веб-интерфейса

После копирования веб-интерфейса откройте в веб-браузере страницу `http://<имя_или_IP-адрес_сервера>/zabbix`.

Перед вами появится страница приветствия, где можно только щелкнуть на кнопке **Next** (Далее). На первой странице может также появиться предупреждение, сообщающее, что часовой пояс не настроен. Соответствующий параметр находится в файле `php.ini`. Определения всех часовых поясов можно найти на официальном веб-сайте PHP: http://www.php.net/manual/ru/timezones.php.

Если текущие настройки PHP вам не известны или вы не знаете, где у вас находится файл `php.ini`, и вам необходима подробная информация о запущенных модулях или текущих настройках, создайте в каталоге Zabbix сценарий в файле с именем, например, `php-info.php` и добавьте в него следующие строки:

```
<?
phpphpinfo();phpinfo(INFO_MODULES);
?>
```

Если после этого открыть в браузере страницу `http://имя_или_IP-адрес_сервера/zabbix/phpinfo.php`, вы получите полную информацию о текущих настройках. На рис. 1.7 показан раздел с описанием предварительных требований, где отсутствующая настройка выделена красным.

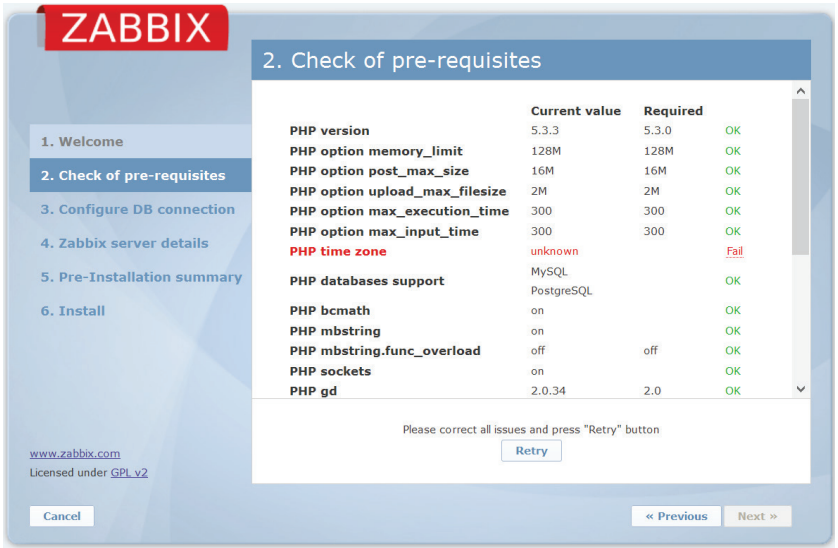


Рис. 1.7 ❖ Информация о текущих настройках

В стандартной установке Red-Hat/CentOS 6.6 необходимо настроить только часовой пояс; но в более старых версиях может потребоваться дополнительно настроить следующие параметры:

```
PHP option post_max_size      8M  16M  Fail
PHP option max_execution_time 30  300  Fail
PHP option max_input_time     60  300  Fail
PHP bcmath                    no       Fail
PHP mbstring                  no       Fail
PHP gd  unknown                   2.0  Fail
PHP gd PNG support            no       Fail
PHP gd JPEG support           no       Fail
PHP gd FreeType support       no       Fail
```

```
PHP xmlwriter                   no      Fail
PHP xmlreader                   no      Fail
```

Большая часть этих параметров определяется в файле `php.ini`. Чтобы настроить их, просто измените содержимое файла `/etc/php.ini`, как показано ниже:

```
[Date]
; Определяет часовой пояс по умолчанию для функций date
; http://www.php.net/manual/ru/datetime.configuration.php#ini.date.timezone
date.timezone = Europe/Moscow

; Максимальный размер данных POST, который PHP сможет принять.
; http://www.php.net/manual/ru/ini.core.php#ini.post-max-size
post_max_size = 16M

; Максимальное время выполнения сценария в секундах.
; http://www.php.net/manual/ru/info.configuration.php#ini.max-execution-time
max_execution_time = 300

; Максимальное время, которое каждый сценарий может потратить
; на парсинг запроса. Желательно ограничить это время на
; промышленных серверах, чтобы исключить неоправданно долгое
; выполнение сценариев.
; Значение по умолчанию: -1 (не ограничено)
; Значение для разработки: 60 (60 секунд)
; Значение для промышленной эксплуатации: 60 (60 секунд)
; http://www.php.net/manual/ru/info.configuration.php#ini.max-input-time
max_input_time = 300
```

Чтобы решить проблему отсутствующих библиотек, установите следующие пакеты:

- `php-xml`;
- `php-bcmath`;
- `php-mbstring`;
- `php-gd`.

Для этого достаточно выполнить команду:

```
# yum install php-xml php-bcmath php-mbstring php-gd
```

В табл. 1.2 приводится полный список требований, предъявляемых веб-интерфейсом Zabbix.

Таблица 1.2 ❖ Требования, предъявляемые веб-интерфейсом Zabbix

Требование	Минимальное значение	Решение
Версия PHP	5.3.0	
`memory_limit`	`128M`	Установить в `php.ini` параметр `memory_limit = 128M`
`post_max_size`	`16M`	Установить в `php.ini` параметр `post_max_size = 16M`
`upload_max_filesize`	`2M`	Установить в `php.ini` параметр `upload_max_filesize = 2M`
`max_execution_time`	`300` (секунд)	Установить в `php.ini` параметр `max_execution_time = 300`

Таблица 1.2 *(окончание)*

Требование	Минимальное значение	Решение
max_input_time	300 (секунд)	Установить в php.ini параметр max_input_time = 300
session_auto_start	Отключить	Установить в php.ini параметр session_auto_start = 0
bcmath		Установить модуль php-bcmath
mbstring		Установить модуль php-mbstring
mbstring.func_overload	Отключить	Установить в php.ini параметр mbstring.func_overload = 0
always_populate_raw_post_data	Присвоить значение -1	Установить в php.ini параметр always_populate_raw_post_data = -1
sockets		Установить модуль php-net-socket
gd		Расширение PHP GD должно поддерживать графические форматы: PNG (--with-png-dir), JPEG (--with-jpeg-dir), а также FreeType 2 (--with-freetype-dir)
libxml		Установить модуль php-xml или php5-dom
xmlwriter		Установить модуль php-xmlwriter
xmlreader		Установить модуль php-xmlreader
ctype		Установить модуль php-ctype
session		Установить модуль php-session
gettext		Установить модуль php-gettext. Начиная с версии 2.2.1 это не является обязательным требованием, но, если его не удовлетворить, могут возникнуть проблемы с интернационализацией

После изменения содержимого файла php.ini или установки модулей расширения PHP необходимо перезапустить службу httpd, чтобы изменения вступили в силу. После удовлетворения всех требований щелкните на кнопке **Next** (Далее), чтобы двинуться дальше. На следующей странице необходимо настроить подключение к базе данных. Для этого достаточно указать в предложенной форме имя пользователя, пароль, IP-адрес или имя хоста и тип сервера баз данных, как показано на рис. 1.8.

Если настройки выполнены без ошибок (в этом можно убедиться, щелкнув на кнопке **Test connection** (Проверить подключение)), можно переходить к следующему шагу. Здесь (рис. 1.9) вам необходимо определить параметры подключения к серверу Zabbix.

Тут не предусмотрена возможность проверки подключения, поэтому желательно вручную проверить доступность сервера Zabbix в сети. В этой форме необходимо заполнить поле **Host** (хост, указав имя хоста или IP-адрес сервера Zabbix). Поскольку мы разворачиваем инфраструктуру на трех разных серверах, требуется указать все параметры и убедиться, что порт сервера Zabbix доступен из сети.

Заполнив форму, щелкните на кнопке **Next** (Далее). После этого мастер настройки выведет сводную информацию о настройках, включающую все конфигурационные параметры. Если все хорошо, щелкните на кнопке **Next** (Далее); в противном случае вернитесь назад и измените ошибочные параметры. После щелчка на кнопке **Next** (Далее) появится конфигурационный файл (в данном примере /usr/share/zabbix/conf/zabbix.conf.php).

Рис. 1.8 ❖ Настройка подключения к базе данных

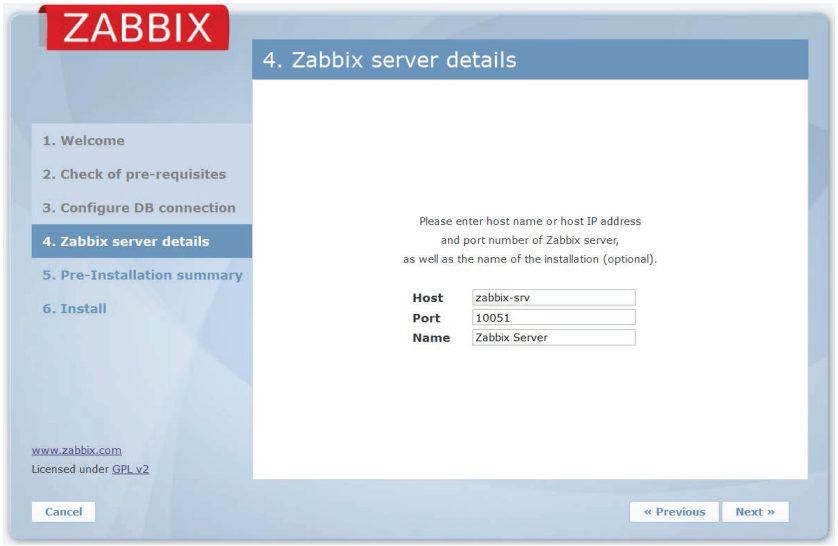


Рис. 1.9 ❖ Настройка подключения к серверу Zabbix

Может так случиться, что вместо уведомления об успехе вы получите сообщение об ошибке, и, вероятнее всего, об ошибке отсутствия привилегий на доступ к каталогу `/usr/share/zabbix/conf`. В этом случае сделайте данный каталог доступным на запись для пользователя `httpd` (с привилегиями которого действует сервер

Apache), хотя бы на время создания файла конфигурации. По окончании описанных шагов веб-интерфейс будет готов к работе, и вы сможете выполнить первый вход в него.

Планирование мощностей с помощью Zabbix

Многие не понимают разницы между планированием мощностей и настройкой производительности. Поэтому хотелось бы уточнить, что настройка производительности – это оптимизация функционирования уже имеющейся системы. Под планированием мощностей подразумевается оценка – что понадобится вашей системе и когда это понадобится. Здесь мы посмотрим, как организовать мониторинг инфраструктуры для достижения этой цели и получить надежную оценку. К сожалению, в одной главе невозможно охватить все аспекты планирования мощностей – для этого потребовалась бы целая книга, тем не менее в этом разделе мы попытаемся взглянуть на Zabbix под разными углами и понять, что может дать нам эта система.

Эффект наблюдателя

Zabbix – отличная система мониторинга, потому что имеет по-настоящему низкие накладные расходы. К сожалению, любая наблюдаемая система тратит часть своих ресурсов на работу агента, который извлекает и оценивает различные показатели работы операционной системы. Поэтому нет ничего странного, что агент вносит небольшие (обычно очень небольшие) накладные расходы. Эти расходы называют **эффектом наблюдателя** (observer effect). Нам остается только согласиться с этим бременем и, понимая, что дополнительная нагрузка вносит некоторые искажения в отбираемые данные, стараться сделать сбор данных как можно более легковесным, управляя процессом мониторинга и нашими собственными проверками.

Выбор параметров для мониторинга

Задача агента Zabbix состоит в том, чтобы периодически собирать данные о работе подконтрольной системы и посылать результаты серверу Zabbix (который собирает и обрабатывает эти результаты). Учитывая все вышесказанное, нам необходимо рассмотреть следующие аспекты.

- Какие данные собирать?
- Как эти данные собирать (какие пути и методы использовать)?
- Как часто производить измерения?

Чтобы ответить на вопрос в первом пункте, необходимо поразмыслить, какие показатели работы хоста нам интересны и какую работу он выполняет; иначе говоря, какие функции на него возложены.

Существует несколько основных показателей работы операционных систем, более или менее стандартизованных. К ним относятся: нагрузка на центральный процессор, процент свободной памяти, статистика использования памяти и файла подкачки, доля процессорного времени, выделяемого процессу, и многие другие. Все они по умолчанию определяются агентом Zabbix.

Наличие набора данных, поддерживаемого по умолчанию, означает, что процедуры их получения оптимизированы и имеют очень низкие накладные расходы.

Все остальные параметры можно разделить по службам, которые должен поддерживать наш сервер.

Для этого с успехом можно использовать шаблоны! (Кроме того, это один из самых эффективных способов сбора информации по типам.)

С точки зрения мониторинга СУБД наибольший интерес представляют:

- все параметры работы операционной системы;
- некоторые отдельные параметры работы СУБД.

К параметрам работы СУБД относятся: количество подключенных пользователей, статистика использования системного кэша, количество операций полного сканирования таблиц и т. д.

Все эти параметры действительно интересны. Они легко поддаются интерполяции и сравнительному анализу с помощью графиков. Графики – сильный инструмент, потому что:

- позволяют легко получить общее представление;
- часто очень удачно вписываются в слайды, сопровождающие наши выступления.

Вернемся к нашему примеру. Прямо сейчас мы уже получаем данные от нашей СУБД и операционной системы, благодаря чему можем сравнивать нагрузку на СУБД и видеть влияние этой нагрузки на работу операционной системы.

А теперь представьте, что основную прибыль бизнес получает от работы вебсайта, интернет-магазина или веб-приложения. Допустим, что нам нужно организовать трехуровневое окружение как наиболее типичное. То есть инфраструктура включает следующие компоненты:

- веб-сервер;
- сервер приложений;
- СУБД.

В реальной жизни система Zabbix чаще всего встраивается в такие окружения. Для успешного мониторинга нам нужно знать все компоненты, влияющие на оказание услуг, обеспечить измерение показателей их работы и сохранение результатов в системе Zabbix. Специалистов, занимающихся системным администрированием, обычно интересуют параметры работы операционной системы, и это нормально. Программистов на Java могут интересовать другие, менее известные параметры, такие как количество потоков выполнения. Свои предпочтения могут иметь администраторы баз данных и другие работники из других отделов.

Это очень важный пункт, поэтому те, кто будет заниматься внедрением Zabbix, должны видеть общую картину и помнить, что для закупки нового оборудования им придется обосновать свои решения перед коммерческим отделом.

Сотрудники коммерческих отделов часто ничего не знают о количестве потоков выполнения, которое способна поддерживать операционная система, – их интересует только удовлетворенность клиента, проблемы, возникающие при их обслуживании, и как много пользователей система может обслуживать одновременно.

Поэтому очень важно уметь общаться с такими людьми на их языке, а это возможно, только имея графики, отражающие изменение понятных им показателей.

Определение базовой оценки

Если взглянуть на инфраструктуру глазами клиента, можно сказать, что если страницы открываются за разумное время, взаимодействие с ней оставляет благоприятные впечатления.

Наша цель в данном случае – доставить удовольствие клиенту и обеспечить надежную работу всей инфраструктуры. Соответственно, нам необходимо измерить следующие характеристики:

- удовлетворенность пользователя (время отклика на запросы веб-страниц);
- параметры работы инфраструктуры, связанные с этим.

Памятуя, что процесс навигации по сайту должен приносить удовольствие, мы должны определить время отклика на действия пользователя и то, как долго пользователь вынужден ждать, пока перед ним откроется очередная веб-страница. Измеренное время отклика можно классифицировать, как описывается ниже:

- **0,2 секунды**: дает ощущение мгновенного отклика – пользователь чувствует, что реакция браузера вызвана его действиями, а не логикой работы сервера;
- **1–2 секунды**: пользователь не выпадает из потока навигации по сайту – он все еще может свободно перемещаться между страницами почти без вынужденных остановок;
- **10 секунд**: пользователь практически наверняка покинет сайт – будучи вынужденным ждать, он, вне всяких сомнений, отвлечется на что-то другое.

Итак, мы определили пороговые значения и можем измерить время отклика в типичном процессе навигации по сайту, а также предусмотреть отправку предупреждения, когда это время превысит две секунды.

Теперь нам нужно увязать это с остальными измеряемыми параметрами: количеством пользователей, подключенных к базе данных, количеством сеансов, открытых сервером приложений, и количеством соединений с базой данных. Нам также нужно увязать все измеряемые параметры со временем отклика и количеством подключенных пользователей, а еще измерить, как система обслуживает страницы в типичном процессе навигации по сайту.

Все это можно определить как базовую оценку функционирования системы при нормальной нагрузке.

Нагрузочное тестирование

Теперь, когда мы определили, что нас интересует, и установили цели, можно двигаться дальше.

Нам нужно знать пределы наших возможностей и, что особенно важно, как система должна откликаться на запросы. Поскольку мы не можем нанять массу народу, которые щелкали бы на ссылки как сумасшедшие, нужно использовать программу, имитирующую подобное поведение. Существует множество открытых

программ, созданных специально для этого. Из доступных альтернатив можно упомянуть программный продукт Siege (https://www.joedog.org/2013/07/siege-3-0-3-url-encoding/).

Программа Seige позволяет имитировать историю посещений страниц в браузере и нагружать сервер запросами. При этом важно помнить, что пользователи, реальные пользователи, никогда не согласовывают своих действий друг с другом. Поэтому требуется ввести задержку между запросами. Помните, что если сайт предусматривает регистрацию пользователей на входе, мы должны использовать базу данных пользователей, потому что сервер приложений кэширует свои объекты, а нам не хотелось бы заниматься измерениями – как лихо он обращается со своим кэшем. Основное правило состоит в том, чтобы создать реальный сценарий навигации по сайту, чтобы вошедшие пользователи могли выходить одним щелчком, без каких-либо задержек.

Хранимые сценарии должны повторяться *x* раз со все увеличивающимся количеством пользователей, а система Zabbix будет сохранять измеренные параметры. В некоторый момент мы превысим первый порог (1–2 секунды на страницу). Можно пойти дальше и увеличивать нагрузку, пока время отклика не достигнет второго порога. Как известно, аппетит приходит во время еды, и я совершенно не удивлюсь, если вы решите пойти еще дальше и будете наращивать нагрузку, пока не обрушите один из компонентов своей инфраструктуры.

Нарисовав график зависимости времени отклика от количества пользователей, можно увидеть, является ли эта зависимость линейной. Вероятнее всего, до определенного предела она будет оставаться линейной. Этот сегмент соответствует режиму нормального функционирования системы. Можно также посмотреть, какие компоненты инфраструктуры вносят задержки, и сделать соответствующие выводы.

Теперь мы точно знаем, чего ожидать от системы и как она будет обслуживать пользователей. Мы сможем узнать, какой компонент вносит наибольший вклад в задержку, и запланировать дальнейшие работы по настройке.

Планирование мощностей можно выполнять без глубокого анализа компонентов, подлежащих оптимизации. Как отмечалось выше, оптимизация производительности и планирование мощностей – это две разные задачи, тесно связанные друг с другом, но разные. Мы можем просто понаблюдать за изменением производительности и запланировать дальнейшее расширение инфраструктуры.

Плановое наращивание мощности аппаратных средств всегда обходится дешевле, чем неожиданное обновление, проводимое в условиях ликвидации аварии.

Можно также выполнить настройку производительности, но при этом необходимо помнить, что между временем, потраченным на настройку, и полученным увеличением производительности тоже есть своя зависимость (как показано на рис. 1.10), и нужно уметь вовремя остановиться, чтобы не тратить слишком много времени на получение незначительной выгоды.

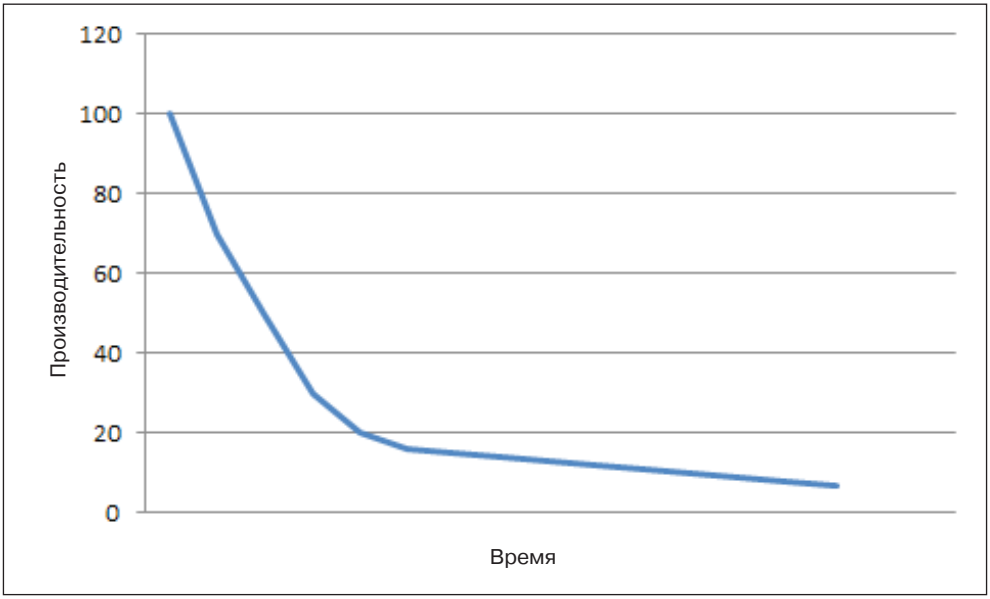


Рис. 1.10 ❖ Зависимость между временем на настройку и полученным увеличением производительности

Прогнозирование тенденций

Одной из важнейших характеристик Zabbix является объем хранимых исторических данных. Она имеет особое значение для прогнозирования. Прогнозирование – не самая простая задача и имеет большое значение для бизнеса, поэтому, просматривая исторические данные, старайтесь увидеть повторяющиеся периоды или своего рода шаблон, который мог бы выразить тенденцию.

Для примера допустим, что интернет-магазину, мониторинг которого мы осуществляем, требуется все больше и больше ресурсов в определенный период года, например перед сезоном отпусков (если мы специализируемся на продаже путевок на курорты). Добавим немного конкретики в наш пример и возьмем такой параметр, как используемое дисковое пространство на сервере. Система Zabbix позволяет экспортировать исторические данные, благодаря чему их можно импортировать в электронную таблицу для дальнейшего анализа. В Excel имеется очень удобная для нас возможность построения графиков. С ее помощью можно без труда построить линию и определить, когда исчерпается доступное дисковое пространство. Для этого сначала нужно построить «точечную диаграмму», на которой также важно отобразить график, отражающий объем дискового пространства. После этого найти математическое уравнение функции, график которой близко совпадает с точечной диаграммой. Существует много видов математических функций на выбор; в данном примере я использовал уравнение линейной зависимости, потому что точечный график явно имеет линейный вид.

Процедура определения математической функции называется аппроксимацией.

Полученный в результате график поможет довольно точно узнать, когда доступное дисковое пространство будет исчерпано.

Теперь должно быть понятно, насколько важно иметь достаточно большой объем исторических данных, учитывающий длительность бизнес-цикла.

Наряду с графиком, отражающим тенденцию, желательно сохранить также формулу и дисперсию, чтобы момент, когда дисковое пространство будет исчерпано при сохранении тенденции, можно было вычислить вручную и узнать точность прогноза.

Полученный мною график показан на рис. 1.11. Из этого графика видно, что если тенденция не изменится, дисковое пространство будет исчерпано 25 июня 2015 г.

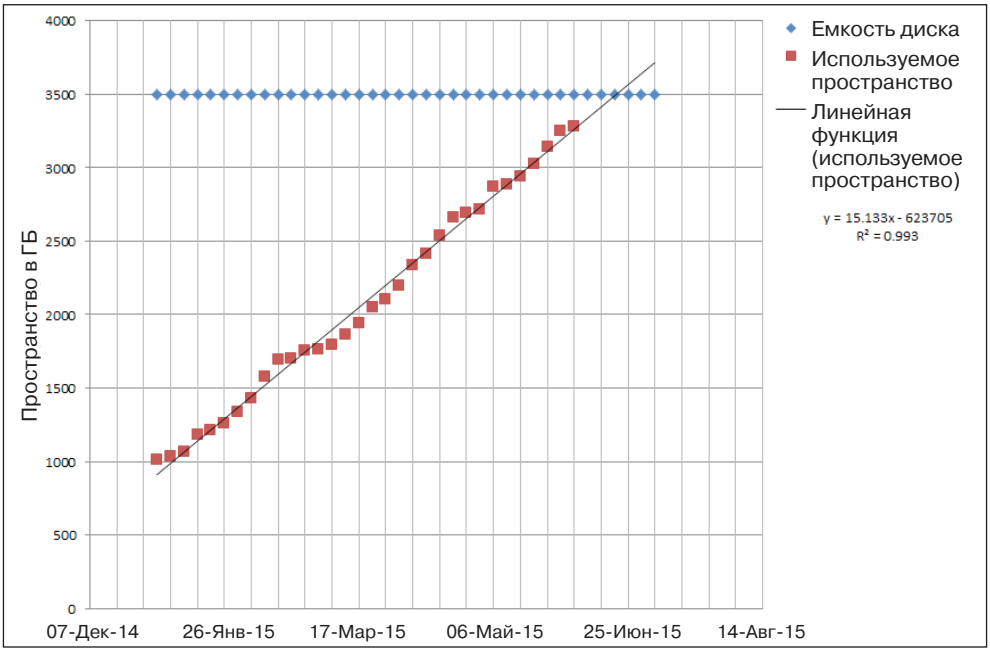


Рис. 1.11 ❖ Прогнозирование исчерпания дискового пространства

В заключение

В этой главе мы выполнили установку и настройку Zabbix в трехуровневом окружении. Такое окружение считается отличной отправной точкой для обработки всех событий, генерируемых в больших и очень больших окружениях.

В следующей главе мы познакомимся с узлами, прокси-серверами и возможными направлениями развития инфраструктуры, которые, как вы увидите сами, являются усовершенствованиями начальной конфигурации, выбранной здесь. Это не означает, что расширения, описываемые в следующей главе, легко осуществить, но все инфраструктурные усовершенствования основываются именно на трехуровневой конфигурации. В следующей главе вы узнаете, как расширить и дополнить эту конфигурацию и как интегрировать в нее возможность распределенного мониторинга. Следующая глава включает также обсуждение такой важной темы, как безопасность распределенного окружения, из которого вы узнаете о возможных рисках, возникающих в таких окружениях.

Глава 2

Распределенный мониторинг

Zabbix – чрезвычайно легковесное приложение мониторинга, способное управлять тысячами элементов данных в конфигурации с единственным сервером. Однако для мониторинга тысяч хостов в сети со сложной топологией или разбросанных географически и связанных линиями с медленной или неустойчивой связью конфигурации с единственным сервером может оказаться недостаточно. Кроме того, необходимость перехода от монолитной конфигурации к распределенной не всегда обусловлена недостаточной производительностью и потому не всегда сводится к выбору между приобретением множества слабых компьютеров или одного большого. На практике часто встречаются сети с сегментами, в которых действуют строгие правила обеспечения безопасности, не допускающие двусторонних взаимодействий между произвольными узлами и, соответственно, не допускающие возможности взаимодействия сервера Zabbix со всеми агентами в таких сегментах. Разным подразделениям одной компании или разным компаниям в одной группе может потребоваться определенная независимость в управлении их сетями. Разные исследовательские лаборатории могут не иметь надежного подключения к сети. В таких случаях требуется организовать накопление контролируемых параметров и их передачу в асинхронном режиме для дальнейшей обработки.

Благодаря своим возможностям распределенного мониторинга Zabbix может с успехом использоваться во всех перечисленных случаях и обеспечить работоспособное решение независимо от причин, будь то недостаточная производительность, сегментация сети, административная независимость или задержка данных из-за ненадежной связи.

Несмотря на то что использование агентов Zabbix вполне можно было бы считать одной из форм распределенного мониторинга, в этой главе мы сосредоточимся на поддержке мониторинга с применением прокси-серверов. Здесь вы узнаете, как установить и настроить прокси-сервер Zabbix.

Мы также коснемся вопросов обеспечения безопасности взаимодействий между сервером и прокси-серверами Zabbix, чтобы к концу этой главы вы имели всю информацию, необходимую для использования средств распределенного мониторинга Zabbix.

Прокси-серверы Zabbix

Прокси-сервер Zabbix – это еще один компонент системы Zabbix, который располагается между основным сервером и агентами Zabbix. Так же как основной сервер, прокси-серверы занимаются сбором информации о работе узлов и ее хранением в специализированной базе данных. Так же как агенты, прокси-серверы не имеют веб-интерфейса и управляются непосредственно центральным сервером. Кроме того, они не занимаются анализом собираемых данных.

Все это делает прокси-сервер Zabbix простым и легковесным инструментом, когда требуется снять часть нагрузки с центрального сервера, упорядочить поток данных мониторинга по сети (например, когда сеть разделена брандмауэрами на сегменты) или в обоих случаях.

На рис. 2.1 изображена простейшая распределенная архитектура, построенная с привлечением прокси-серверов Zabbix.

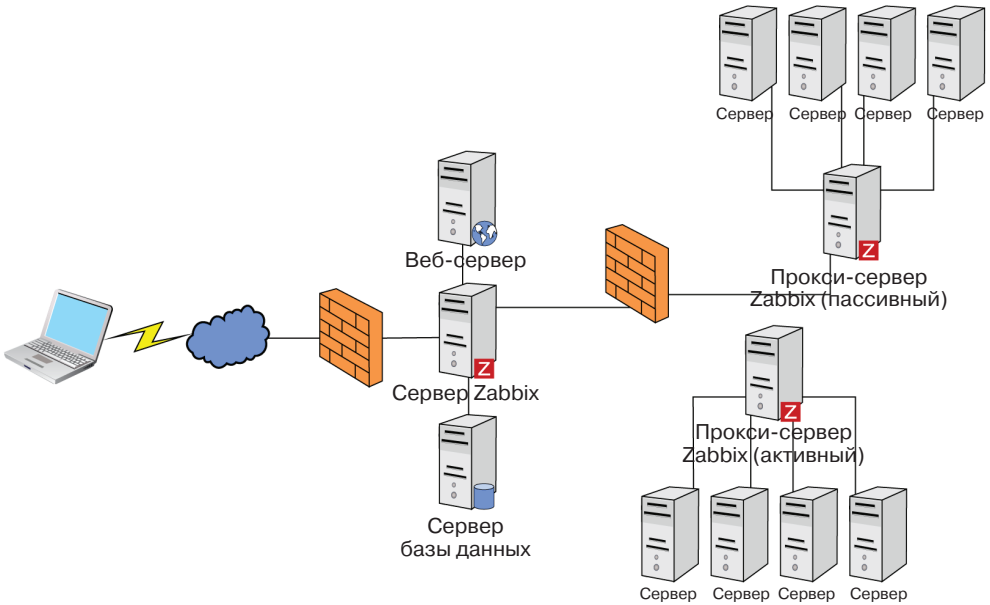


Рис. 2.1 ❖ Архитектура Zabbix с прокси-серверами

По определению прокси-сервер Zabbix должен выполняться на выделенной машине, отдельно от основного сервера. Главная задача прокси-сервера – сбор данных. Он не имеет пользовательского интерфейса и не выполняет никаких сложных запросов или вычислений, поэтому для его установки не требуется машина с мощным процессором или жестким диском, обладающим высокой пропускной способностью. В действительности маломощный компьютер часто оказывается лучшим выбором; прокси-сервер должен быть легковесным – не только чтобы отражать простоту программного компонента, но также для простоты расширения

и распределения архитектуры мониторинга без необходимости вложения больших средств.

Исключением из правила, описанного выше, являются случаи, когда за одним прокси-сервером закрепляются сотни хостов с тысячами параметров, подлежащих мониторингу. Однако в таких ситуациях вместо установки более мощного компьютера часто дешевле обходится разделение хостов на группы и закрепление их за разными небольшими прокси-серверами. Такое решение выглядит предпочтительнее не только потому, что распределяет и выравнивает нагрузку, но также уменьшает риск потери больших объемов данных из-за выхода из строя прокси-сервера. В качестве прокси-серверов Zabbix можно также использовать маленькие и легковесные встраиваемые машины. Они дешевле, проще развертываются, надежнее и весьма экономичны в смысле потребления электроэнергии. Они имеют идеальные характеристики для любого решения мониторинга, особенно при наличии ограничений по габаритам. Но есть другая сторона медали: если сеть разделена на сегменты по географическому положению, желательно, чтобы прокси-серверы обладали достаточно емкой собственной базой данных, так как в таких сетях связь с центральным сервером может пропадать на длительное время. Прокси-сервер должен иметь возможность хранить данные в течение таких периодов, но и в этом случае, если прокси-сервер выйдет из строя, это может стать серьезной проблемой.

Определяя период времени, в течение которого прокси-сервер должен хранить данные из-за отсутствия связи с центральным сервером, необходимо учитывать два фактора: количество подконтрольных хостов и количество элементов данных, которые прокси-сервер должен хранить в своей локальной базе данных. Нетрудно догадаться, что размышления над этой проблемой приведут к необходимости выбора базы данных. Если прокси-сервер находится в локальной сети, предпочтение следует отдавать легковесным и производительным базам данных, таким как SQLite3; в противном случае желательно выбирать другие типы баз данных, способные хранить информацию продолжительное время и более надежные, чем MySQL или PostgreSQL.

Развертывание прокси-сервера Zabbix

Прокси-сервер Zabbix компилируется вместе с основным сервером, если добавить параметр компиляции `--enable-proxy`. Он может использовать в качестве локальной любую базу данных, как и основной сервер, но если не указать существующую базу данных, по умолчанию создается база данных SQLite. Если вы намерены использовать ее, просто добавьте параметр `--with-sqlite3`.

Вообще, прокси-серверы должны оставаться максимально простыми и легковесными. Разумеется, это утверждение верно, только если архитектура сети допускает такое решение. База данных прокси-сервера предназначена только для хранения конфигурации и данных мониторинга, которые при обычных условиях практически немедленно передаются центральному серверу. Создание отдельной полноценной базы данных для этих целей – практически всегда излишество. По-

этому если у вас нет определенных требований, выбор SQLite даст лучший баланс между производительностью и простотой управления.

Если в процессе развертывания Zabbix вы не скомпилировали прокси-сервер, просто запустите утилиту `configure` еще раз, добавив параметры, необходимые для компиляции прокси:

```
$ ./configure --enable-proxy --enable-static --with-sqlite3 --with-netsnmp --with-libcurl
--with-ssh2 --with-openipmi
```

Чтобы выполнить статическую сборку прокси-сервера, необходимо иметь статические версии всех необходимых библиотек. Сценарий `configure` не проверяет их наличия в системе.

и выполните компиляцию командой:

```
$ make
```

При этом будет скомпилирован основной сервер, поэтому не забудьте выполнить команду `make install` или скопируйте новый выполняемый файл сервера Zabbix поверх существующего.

Чтобы установить прокси-сервер, достаточно скопировать на целевую машину конфигурационный и выполняемый файлы прокси-сервера. В примере ниже переменная `$PREFIX` должна хранить тот же путь, который был указан в параметре команды `configure` (по умолчанию `/usr/local`):

```
# cp src/zabbix_proxy/zabbix_proxy $PREFIX/sbin/zabbix_proxy
# cp conf/zabbix_proxy.conf $PREFIX/etc/zabbix_proxy.conf
```

Далее нужно добавить актуальную информацию в конфигурационный файл. Параметры по умолчанию хорошо подходят в большинстве случаев, но следующие обязательно требуется изменить, чтобы они отражали ваши требования и особенности строения сети:

```
ProxyMode=0
```

Этот параметр означает, что прокси-сервер действует в активном режиме. Помните, что количество настроенных ловушек (трапперов) на центральном сервере Zabbix не должно быть меньше количества прокси-серверов. Присвойте этому параметру значение `1`, если необходимо, чтобы прокси-сервер действовал в пассивном режиме. Подробнее о режимах работы прокси-серверов рассказывается в разделе «Движение данных мониторинга в системе Zabbix».

Следующий параметр

```
Server=n.n.n.n
```

определяет IP-адрес центрального сервера или узла Zabbix, которому прокси-сервер должен передавать накопленные данные.

Параметр

```
Hostname=Zabbix proxy
```

определяет уникальное (чувствительное к регистру символов) имя, которое будет использовано в конфигурации центрального сервера Zabbix для ссылки на прокси-сервер.

```
LogFile=/tmp/zabbix_proxy.log
LogFileSize=1
DebugLevel=2
```

Если в роли прокси-серверов используются небольшие встраиваемые машины, в их распоряжении может иметься лишь небольшой объем дискового пространства. В этом случае рекомендуется закомментировать все параметры, имеющие отношение к журналированию, и настроить в `syslog` отправку журнала прокси-сервера на другой сервер в сети.

```
# DBHost=
# DBSchema=
# DBUser=
# DBPassword=
# DBSocket=
# DBPort=
```

Теперь нужно создать базу данных SQLite. Сделать это можно с помощью следующих команд:

```
$ mkdir -p /var/lib/sqlite/
$ sqlite3 /var/lib/sqlite/zabbix.db < /usr/share/doc/zabbix-proxy-sqlite3-2.4.4/create/
schema.sql
```

После этого в параметре `DBName` укажите полный путь к базе данных SQLite:

```
DBName=/var/lib/sqlite/zabbix.db
```

Прокси-сервер автоматически будет заполнять и использовать базу данных SQLite. Но если вы пользуетесь внешней базой данных, настройте соответствующие параметры, управляющие подключением к ней.

```
ProxyOfflineBuffer=1
```

Этот параметр определяет количество часов, в течение которых прокси-сервер должен хранить данные мониторинга в случае потери связи с центральным сервером Zabbix. Данные, хранящиеся дольше этого периода, будут автоматически удаляться. Вы можете удвоить или утроить это значение, если известно, что центральный сервер и прокси-сервер связаны ненадежным соединением.

```
CacheSize=8M
```

Этот параметр определяет размер конфигурационного кэша. Увеличьте это значение, если прокси-сервер обслуживает большое количество хостов и элементов данных.

Команды управления прокси-сервером Zabbix во время выполнения

Прокси-сервер поддерживает набор команд, с помощью которых можно изменять параметры выполнения прямо во время его работы. Эти команды удобно использовать для разрешения проблем и управления прокси-серверами Zabbix.

Следующая команда заставит прокси-сервер обновить конфигурационный кэш, запросив его с центрального сервера Zabbix:

```
$ zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R config_cache_reload
```

Она объявит недействительным конфигурационный кэш на стороне прокси-сервера и вынудит его запросить текущую конфигурацию с центрального сервера Zabbix.

Имеется также возможность увеличивать или уменьшать уровень серьезности журнальных записей. Делается это с помощью команд `log_level_increase` и `log_level_decrease`:

```
$ zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase
```

Данная команда увеличит уровень серьезности журнальных записей, оставляемых прокси-сервером. Эти команды принимают дополнительный параметр, в котором можно передать идентификатор процесса (PID), тип процесса или тип процесса и номер. Например:

- увеличить уровень серьезности журнальных записей для третьего процесса-регистратора (poller):

  ```
  $ zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase=poller,3
  ```

- увеличить уровень серьезности журнальных записей для процесса с PID = 27425:

  ```
  $ zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase=27425
  ```

- увеличить уровень серьезности журнальных записей для icmp-пробника (icmp pinger) или любого другого процесса:

  ```
  $ zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_increase="icmp
  pinger"

  zabbix_proxy [28064]: command sent successfully

  $ zabbix_proxy -c /usr/local/etc/zabbix_proxy.conf -R log_level_decrease="icmp
  pinger"

  zabbix_proxy [28070]: command sent successfully
  ```

Результаты изменений можно тут же обнаружить в файле журнала:

```
28049:20150412:021435.841 log level has been increased to 4 (debug)
28049:20150412:021443.129 Got signal [signal:10(SIGUSR1),sender_pid:28034,sender_uid:501,
value_int:770(0x00000302)].
28049:20150412:021443.129 log level has been decreased to 3 (warning)
```

Развертывание прокси-сервера Zabbix из RPM-пакета

Развертывание прокси-сервера Zabbix из RPM-пакета выполняется очень просто. Для этого требуется выполнить меньше шагов, потому что компания Zabbix включает в собранные пакеты уже готовый прокси-сервер.

Прежде всего необходимо подключить официальный репозиторий Zabbix следующей командой, которая должна запускаться с привилегиями root:

```
$ rpm -ivh http://repo.zabbix.com/zabbix/2.4/rhel/6/x86_64/zabbix-2.4.4-1.el6.x86_64.rpm
```

После этого можно просмотреть список доступных пакетов `zabbix-proxy`:

```
$ yum search zabbix-proxy

============== N/S Matched: zabbix-proxy ===============
zabbix-proxy.x86_64 : Zabbix Proxy common files
zabbix-proxy-mysql.x86_64 : Zabbix proxy compiled to use MySQL
zabbix-proxy-pgsql.x86_64 : Zabbix proxy compiled to use PostgreSQL
zabbix-proxy-sqlite3.x86_64 : Zabbix proxy compiled to use SQLite3
```

Выполнение команды сопровождается выводом списка всех доступных пакетов `zabbix-proxy`; вам останется только выбрать нужный и установить его:

```
$ yum install zabbix-proxy-sqlite3
```

После установки прокси-сервер можно запустить командой:

```
$ service zabbix-proxy start

Starting Zabbix proxy:                                 [ OK ]
```

Не забудьте также настроить автоматический запуск прокси-сервера в момент загрузки компьютера командой:

```
$ chkconfig zabbix-proxy on
```

Вот и все. Если вы используете `iptables`, добавьте правило, пропускающее входящий трафик на порт `10051` (стандартный порт прокси-сервера Zabbix) или, точнее, на порт, указанный в конфигурационном файле:

```
ListenPort=10051
```

Для этого откройте в редакторе конфигурационный файл `/etc/sysconfig/iptables` и добавьте в самое начало следующую строку:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 10051 -j ACCEPT
```

После этого следует перезапустить брандмауэр командой (с привилегиями root):

```
$ service iptables restart
```

После запуска прокси-сервера в файле журнала `/var/log/zabbix/zabbix_proxy.log` должны появиться следующие записи:

```
$ tail -n 40 /var/log/zabbix/zabbix_proxy.log
62521:20150411:003816.801 **** Enabled features ****
62521:20150411:003816.801 SNMP monitoring:       YES
62521:20150411:003816.801 IPMI monitoring:       YES
62521:20150411:003816.801 WEB monitoring:        YES
62521:20150411:003816.801 VMware monitoring:     YES
62521:20150411:003816.801 ODBC:                  YES
62521:20150411:003816.801 SSH2 support:          YES
62521:20150411:003816.801 IPv6 support:          YES
62521:20150411:003816.801 **************************
62521:20150411:003816.801 using configuration file: /etc/zabbix/zabbix_proxy.conf
```

Как вы могли заметить, по умолчанию конфигурация прокси-сервера хранится в файле `/etc/zabbix/zabbix_proxy.conf`.

Единственное, что осталось сделать, – сообщить центральному серверу о существовании прокси и добавить объекты для мониторинга. Все эти задачи решаются с помощью веб-интерфейса Zabbix, в котором следует открыть раздел **Admin** ⇨ **Proxies** (Администрирование ⇨ Прокси) и щелкнуть на кнопке **Create** (Создать). На рис. 2.2 показано, как выглядит форма создания прокси.

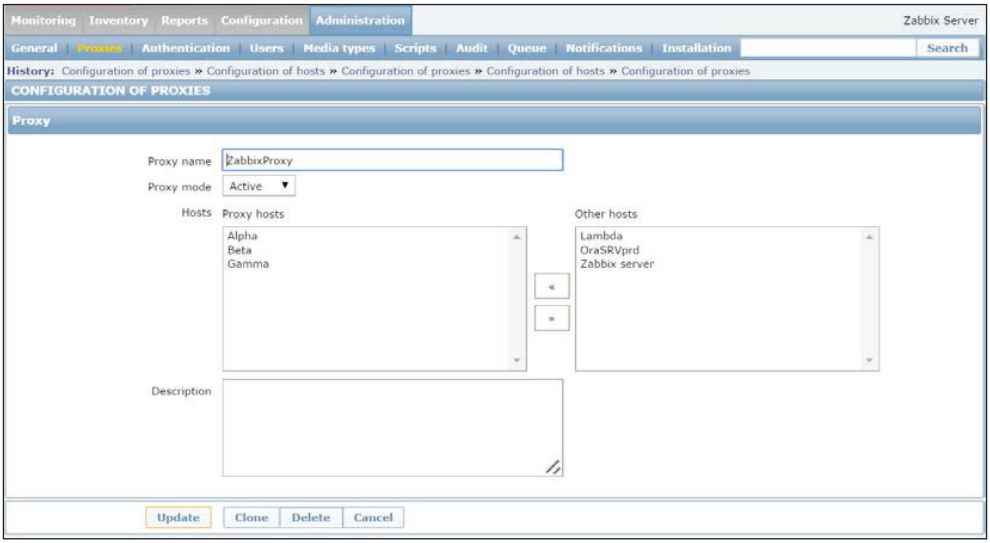


Рис. 2.2 ❖ Настройка нового прокси-сервера

В поле **Proxy name** (Имя прокси) следует указать то же имя, что использовано в конфигурации прокси-сервера, в данном случае `ZabbixProxy`. Это имя можно быстро узнать с помощью команды:

```
$ grep Hostname= /etc/zabbix/zabbix_proxy.conf
# Hostname=
Hostname=ZabbixProxy
```

Обратите внимание, что для прокси-сервера, действующего в активном режиме (значение **Active** в поле **Proxy mode** (Режим прокси)), достаточно указать только его имя, установленное в `zabbix_proxy.conf`. Прокси-сервер сам возьмет на себя труд подключиться к центральному серверу. С другой стороны, если прокси-сервер действует в пассивном режиме, необходимо в поле **IP address** (IP-адрес) указать IP-адрес прокси-сервера или его имя хоста, чтобы центральный сервер мог соединиться с ним (см. рис. 2.3).

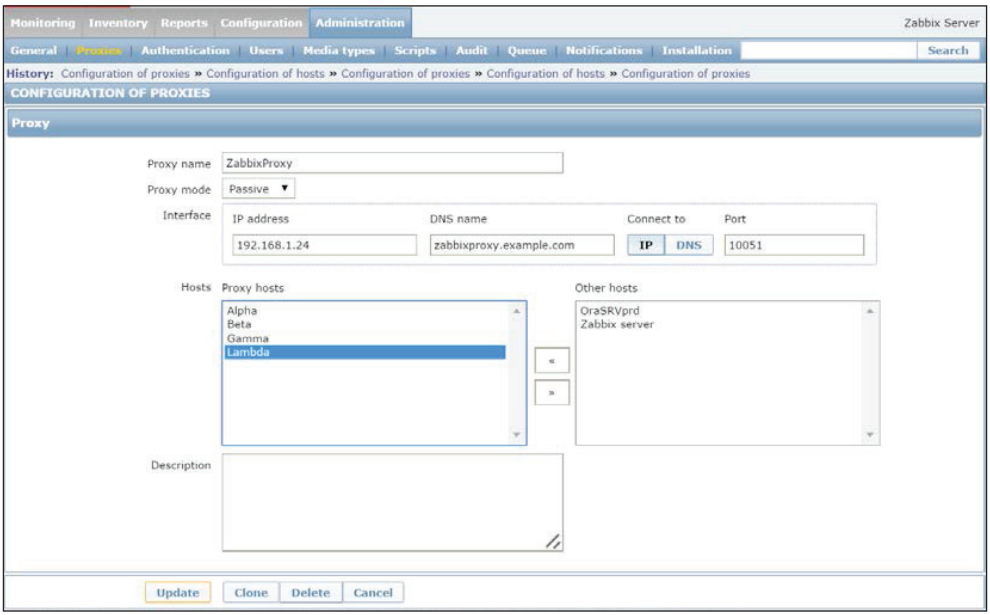


Рис. 2.3 ❖ Дополнительные настройки для прокси-сервера, действующего в активном режиме

Дополнительные подробности приводятся в разделе «Движение данных мониторинга через прокси-серверы». Во время добавления нового или изменения настроек существующего прокси-сервера не требуется обязательно закреплять за ним контролируемые хосты. Это можно сделать в экране конфигурирования хостов, как показано на рис. 2.4.

Одно из достоинств прокси-серверов – они не требуют много настроек или какого-то особого обслуживания; после развертывания и закрепления за ними группы хостов все остальные операции по мониторингу они выполняют автоматически. Достаточно периодически проверять число значений в поле **Required performance** (Требуемая производительность), как показано на рис. 2.5, извлекаемых в секунду.

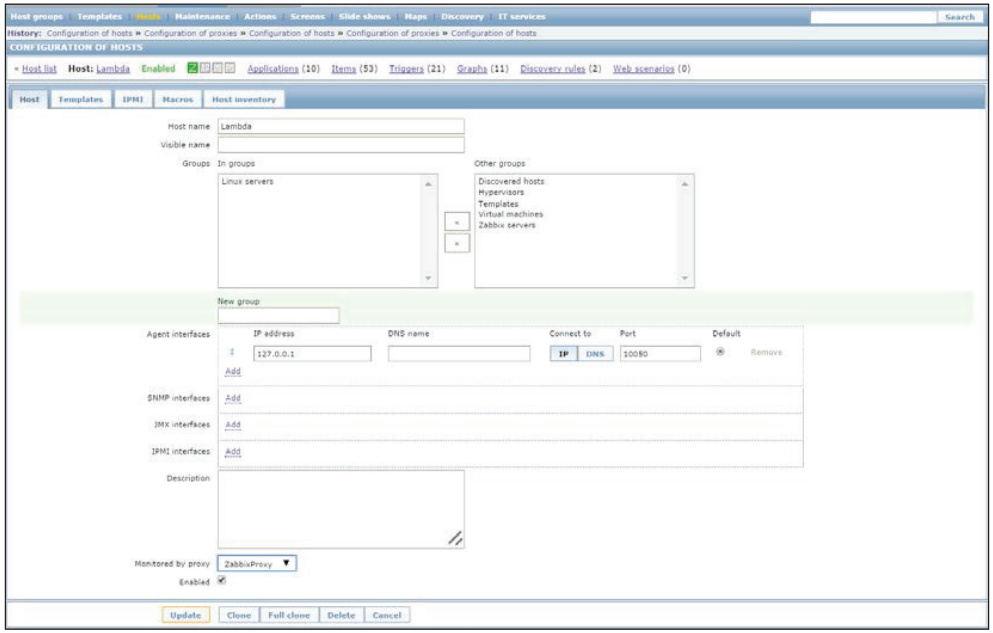


Рис. 2.4 ❖ Конфигурирование списков контролируемых хостов

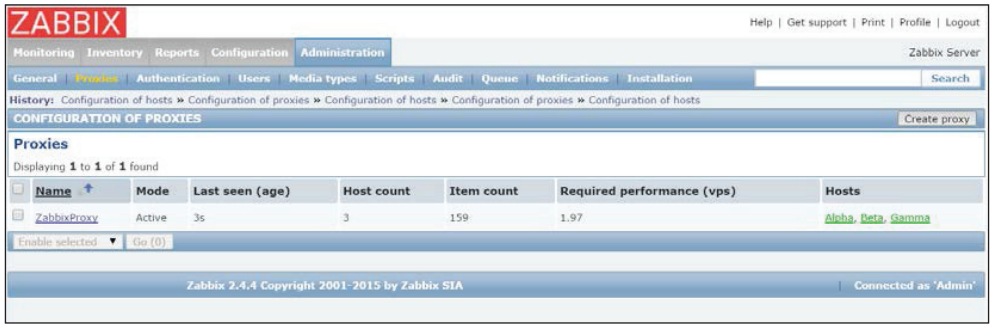


Рис. 2.5 ❖ Конфигурация прокси-сервера

Значение **vps** (values per second – значений в секунду) – это количество параметров, которое один сервер или прокси-сервер Zabbix должен извлекать каждую секунду. Здесь отображается среднее число, которое зависит от количества элементов данных и частоты опроса каждого элемента. Чем больше число, тем мощнее должен быть компьютер под сервером.

В зависимости от конфигурации аппаратуры может потребоваться перераспределить хосты между прокси-серверами или добавить новые прокси, если обнаружится падение производительности в паре с высоким значением **VPS**.

Использование других баз данных с прокси-серверами

Начиная с версии Zabbix 2.4 поддержка узлов (nodes) прекращена, и теперь распределенный мониторинг возможен только с применением прокси-серверов Zabbix, которые играют по-настоящему важную роль. Кроме того, благодаря развертыванию прокси-серверов в географически распределенных фрагментах сети инфраструктура стала более устойчивой к отключениям электричества. Учитывая сказанное, необходимо рассмотреть проблему выбора базы данных для использования на удаленных прокси-серверах.

База данных SQLite3 отлично подходит для использования в монолитных и легковесных конфигурациях, но нужно признать, что SQLite3 плохо подходит в случаях, если прокси должен иметь возможность хранить существенные объемы данных мониторинга:

- механизм атомарной блокировки в SQLite3 пока не отличается высокой надежностью;
- SQLite3 страдает падением производительности при записи больших объемов данных;
- SQLite3 не реализует никаких механизмов аутентификации.

Кроме того что SQLite3 не реализует никаких механизмов аутентификации, файлы базы данных создаются со стандартным значением `umask`, из-за чего они доступны для чтения всем желающим. Одним словом, это не самый лучший выбор для конфигураций с высокой нагрузкой.

Ниже приводится пример базы данных `sqlite3`, демонстрирующий, как получить доступ к ней, используя стороннюю учетную запись:

```
$ ls -la /tmp/zabbix_proxy.db
-rw-r--r--. 1 zabbix zabbix 867328 Apr 12 09:52 /tmp/zabbix_proxy.db
$ su - adv
[adv@localhost ~]$ sqlite3 /tmp/zabbix_proxy.db
SQLite version 3.6.20
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

По этим причинам для всех важных прокси-серверов рекомендуется использовать другую базу данных. Далее демонстрируется настройка поддержки известной базы данных MySQL.

Прежде всего необходимо настроить поддержку MySQL, если вы устанавливаете систему из исходных кодов:

```
$ ./configure --enable-proxy --enable-static --with-mysql --with-net-snmp --with-libcurl
--with-ssh2 --with-openipmi
```

и собрать как обычно:

```
$ make
```

Если вы устанавливаете систему из пакетов rpm, просто выполните следующую команду (с привилегиями root):

```
$ yum install zabbix-proxy-mysql
```

После этого запустите СУБД MySQL и создайте базу данных для прокси-сервера:

```
$ mysql -uroot -p<пароль>
$ create database zabbix_proxy character set utf8 collate utf8_bin;
$ grant all privileges on zabbix_proxy.* to zabbix@localhost identified by '<пароль>';
$ quit;
$ mysql -uzabbix -p<пароль> zabbix_proxy < database/mysql/schema.sql
```

Если установка системы выполнялась из пакетов rpm, последняя команда в предыдущем блоке должна быть такой:

```
$ mysql -uzabbix -p<пароль> zabbix_proxy < /usr/share/doc/zabbix-proxy-mysql-2.4.4/create/
schema.sql/schema.sql
```

Теперь отредактируйте следующие параметры в файле `zabbix_proxy.conf`:

```
DBName=zabbix_proxy
DBUser=zabbix
DBPassword=<пароль>
```

Обратите внимание, что в данном случае нет необходимости определять параметр `DBHost` для MySQL.

В заключение запустите прокси-сервер:

```
$ service zabbix-proxy start
Starting Zabbix proxy:                          [ OK ]
```

Движение данных мониторинга в системе Zabbix

Прежде чем объяснить, как данные мониторинга движутся через прокси-серверы, важно иметь хотя бы общее представление о движении данных в стандартной системе Zabbix.

Всего существуют четыре вида источников, которые могут поставлять данные серверу Zabbix:

- агент Zabbix;
- утилита командной строки Zabbix Sender;
- нестандартные агенты от сторонних производителей;
- прокси-сервер Zabbix.

На рис. 2.6 изображена упрощенная диаграмма движения данных.

Не забывайте, что это лишь упрощенная версия полной схемы потоков данных, и на ней многочисленные мелкие компоненты объединены в один блок **Прочие**. Итак, слева на рис. 2.6 изображены все четыре вида источников данных, а справа – веб-интерфейс Zabbix (**HTTP**) и, конечно же, база данных, где хранятся все данные. Теперь, в следующем разделе, посмотрим, как движутся данные мониторинга через прокси-сервер Zabbix.

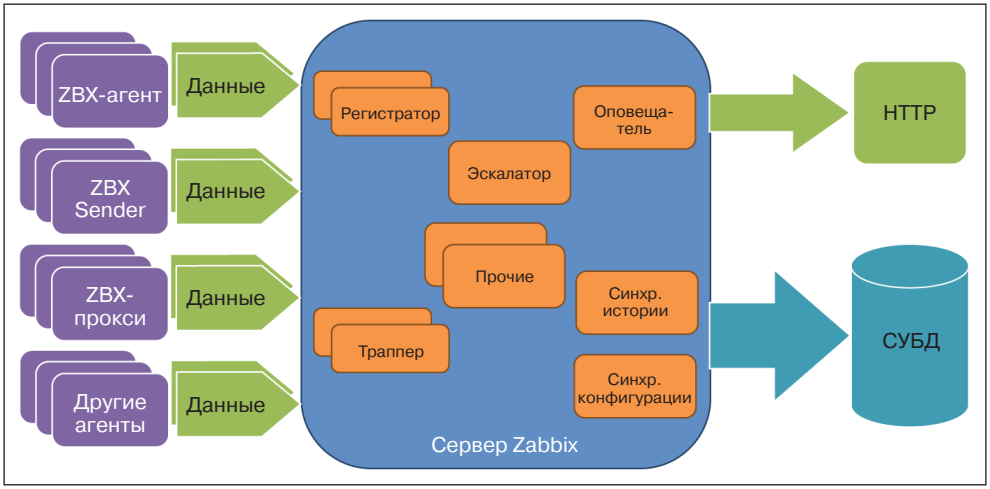


Рис. 2.6 ❖ Движение данных мониторинга в системе Zabbix

Движение данных мониторинга через прокси-серверы

Прокси-серверы Zabbix способны действовать в двух режимах, активном и пассивном. В активном режиме, который устанавливается по умолчанию, прокси сам инициирует все соединения с сервером Zabbix, необходимые для извлечения конфигурационной информации об объектах мониторинга и отправки полученных данных для последующей обработки. В конфигурационном файле прокси имеются соответствующие параметры, позволяющие настроить частоту выполнения этих двух операций:

```
ConfigFrequency=3600

DataSenderFrequency=1
```

Оба значения в этом примере выражены в секундах. На стороне сервера Zabbix, в файле `zabbix_server.conf`, необходимо также установить в параметре `StartTrappers` значение, превышающее число всех активных прокси. Процессы ловушек (траппероров) управляют информацией, поступающей от прокси-серверов, узлов (nodes) и других элементов, настроенных на работу в активном режиме. Сервер будет порождать дополнительные процессы по мере необходимости, но желательно изначально запустить достаточное количество процессов, чтобы потом не тратить времени на их запуск.

Вернемся к прокси-серверу. В его конфигурационном файле можно также установить параметр `HeartbeatFrequency`, чтобы заставить его подключаться к центральному серверу через предопределенное количество секунд, даже если нет данных для отправки. После этого можно проверять доступность прокси с помощью следующего элемента:

```
zabbix[proxy, "proxy name", lastaccess]
```

где `proxy name` – уникальный идентификатор, присвоенный прокси-серверу во время развертывания.

Элемент возвращает количество секунд, прошедших с момента последнего подключения прокси, которое затем можно использовать в тех или иных функциях. Чтобы определить оптимальную частоту проверочных соединений прокси-сервера с основным сервером, сначала нужно выяснить, на какой период допустимо потерять контакт с ним, прежде чем сгенерировать предупреждение, и затем установить интервал между проверочными соединениями, чуть меньше половины этого периода. Например, если допускается потеря связи с прокси-сервером не более чем на 5 минут (300 секунд), установите интервал между проверочными соединениями 120 секунд и сравнивайте количество секунд, прошедших с момента последнего подключения прокси с числом 300. На рис. 2.7 изображена диаграмма, в точности отражающая сказанное выше.

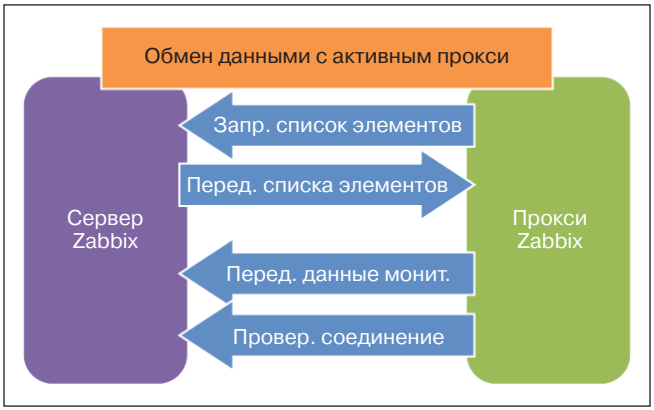


Рис. 2.7 ❖ Обмен данными с активным прокси-сервером

Активный прокси-сервер эффективнее разгружает основной сервер, поскольку последнему остается только «сидеть и ждать» запросов на получение обновленной конфигурации или новых данных мониторинга. Но прокси-серверы часто развертываются для мониторинга закрытых сетей, сегментов со строгими ограничениями на исходящий трафик. В таких случаях бывает трудно получить разрешение на инициацию соединений со стороны прокси-сервера. Но это не только вопрос политики; ограничения могут накладываться по целому ряду других, весьма веских причин. С другой стороны, часто проще и безопаснее инициировать соединения с закрытыми сетями извне. В таких случаях предпочтительнее использовать пассивный прокси-сервер.

Пассивный режим работы прокси-сервера является практически зеркальным отражением активного режима. На этот раз уже центральный сервер должен периодически соединяться с прокси, чтобы послать изменения в конфигурации и запросить данные мониторинга. Чтобы перевести прокси-сервер в пассивный

режим, в его конфигурационном файле достаточно установить единственный параметр `ProxyMode=1`. Но на стороне центрального сервера требуется определить три дополнительных параметра:

- `StartProxyPollers` – определяет количество процессов, выделенных для управления пассивными прокси-серверами, и должно совпадать с количеством развернутых пассивных прокси-серверов;
- `ProxyConfigFrequency` – определяет интервал времени в секундах, через который центральный сервер будет посылать пассивному прокси-серверу измененную конфигурацию;
- `ProxyDataFrequency` – определяет интервал времени в секундах между двумя последовательными запросами к прокси-серверу на отправку данных мониторинга.

В остальном пассивный и активный режимы работы прокси-серверов ничем не отличаются, как показано на рис. 2.8: для проверки доступности пассивного прокси-сервера все так же можно использовать элемент `zabbix[proxy, "proxy name", lastaccess]`.

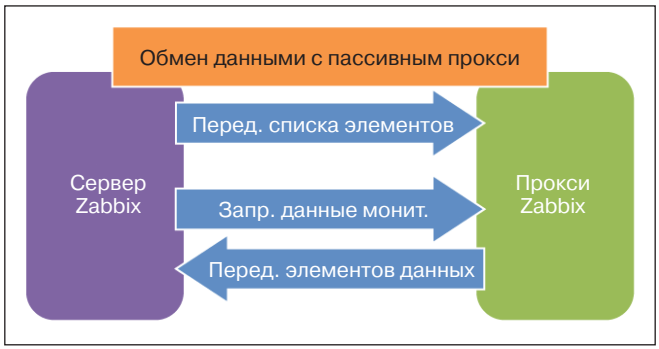


Рис. 2.8 ❖ Обмен данными с пассивным прокси-сервером

За счет небольшого увеличения нагрузки на центральный сервер пассивные прокси-серверы позволяют организовать мониторинг закрытых сетей и их сегментов. Так или иначе, в зависимости от правил, действующих в конкретных сетях, можно одновременно использовать и активные, и пассивные прокси. Это позволяет расширить возможности решения мониторинга в плане достижения всех сегментов сети и его способности обрабатывать большое количество объектов, сохранив при этом архитектуру максимально простой для управления с мощным центральным ядром и простой, легковесной, но достаточно эффективной периферией.

Мониторинг прокси-серверов Zabbix

Прокси-серверы – единственный компонент, способный взять на себя часть нагрузки сервера Zabbix, а также единственный способ учесть топологию сети, поэтому мы должны с особым вниманием относиться к ним.

Мы уже видели, как определить элемент для их мониторинга и организовать контрольные запросы на стороне активных прокси, но этого недостаточно.

Существует еще несколько полезных элементов, содержащихся в шаблоне `Template App Zabbix Proxy`. Этот шаблон обязательно нужно знать и использовать.

К сожалению, нет элемента, который позволил бы узнать, как много элементов осталось в очереди прокси для передачи центральному серверу.

Это, пожалуй, самый очевидный и важный параметр. Проблему его отсутствия в стандартном наборе можно решить с помощью следующего запроса к базе данных прокси-сервера:

```
SELECT
  ((SELECT MAX(proxy_history.id)
    FROM proxy_history)-nextid)
FROM ids
WHERE field_name='history_lastid'
```

Этот запрос вернет количество элементов, которые прокси-сервер все еще должен отправить серверу Zabbix. Самый простой способ выполнить этот запрос при использовании базы данных SQLite3 – добавить следующий пользовательский параметр `UserParameter` в конфигурацию прокси:

```
UserParameter=zabbix.proxy.items.sync.remaining,/usr/bin/sqlite3/путь/к/базе/данных/sqlite/
"SELECT ((SELECT MAX(proxy_history.id) FROM proxy_history)-nextid) FROM ids WHERE field_
name='history_lastid'" 2>&1
```

Если для прокси выбрана более надежная база данных, например MySQL, тогда `UserParameter` следует определить иначе:

```
UserParameter=zabbix.proxy.items.sync.remaining, mysql -u <имя пользователя> -p'<пароль>'
<имя БД> -e 'SELECT ((SELECT MAX(proxy_history.id) FROM proxy_history)-nextid) FROM ids
WHERE field_name=history_lastid' 2>&1
```

Теперь осталось лишь настроить элемент на стороне сервера Zabbix, с соответствующим триггером, который будет следить за освобождением очереди на прокси-сервере. Форма создания этого элемента показана на рис. 2.9.

Ниже приводится пример триггера, который можно было бы связать с этим элементом:

```
{Hostname:zabbix.proxy.items.sync.remaining.min(10m)}>10000
```

Этот триггер будет срабатывать, когда количество неотправленных элементов в очереди достигнет 10 000, но вам наверняка может понадобиться скорректировать это число в зависимости от количества хостов, опрашиваемых прокси-сервером, и количества элементов, извлекаемых им.

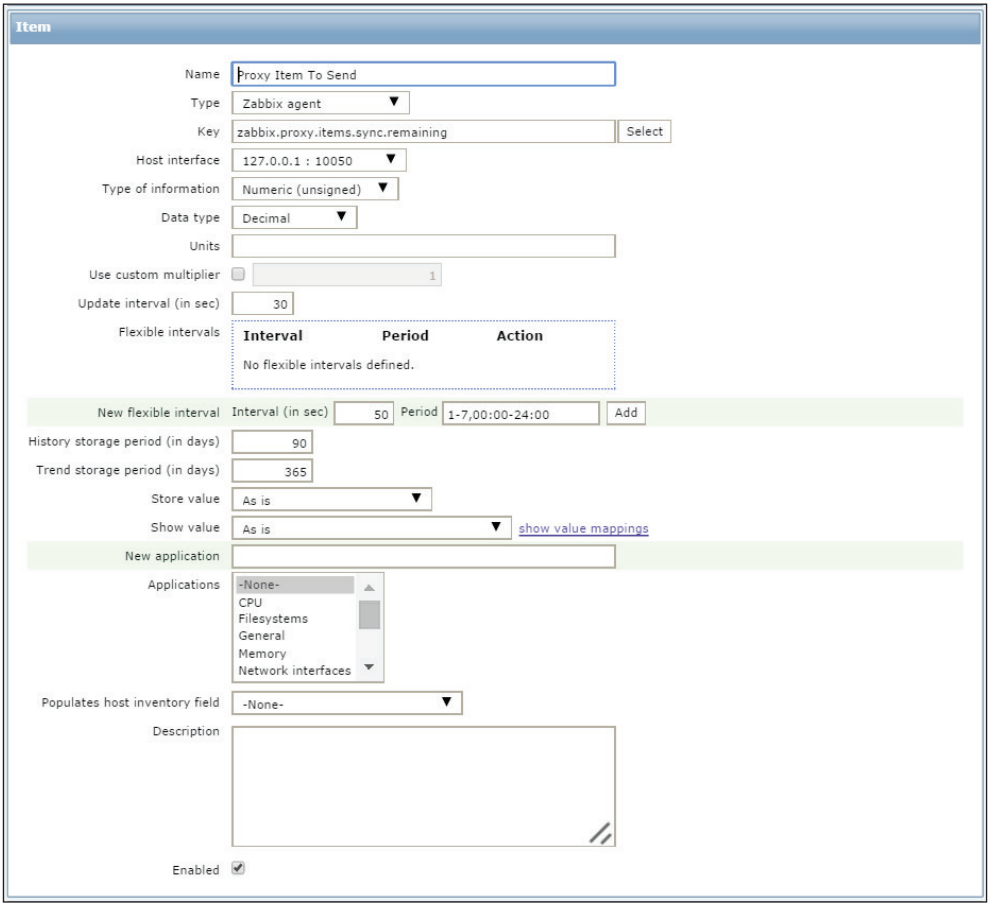


Рис. 2.9 ❖ Определение элементов для слежения за освобождением очереди на прокси-сервере

Вопросы безопасности

Одним из немногих недостатков всей архитектуры Zabbix является отсутствие встроенных средств поддержки безопасности на уровне протокола. Несмотря на то что веб-интерфейс и Zabbix API можно защитить стандартными средствами SSL, не существует простого и стандартного способа защитить взаимодействия между агентами и сервером, между прокси и сервером или между узлами. Не существует также стандартных способов аутентификации сообщений (отправитель – действительно тот, за кого себя выдает), проверки целостности (данные не были изменены на полпути) или обеспечения конфиденциальности (никто другой не сможет прочитать или понять данных).

Обращавшие внимание на конфигурацию агентов, прокси-серверов и узлов могли заметить, что для взаимодействий друг с другом компонентам Zabbix достаточно знать только IP-адреса. Никакой аутентификации не предусмотрено, а IP-адрес не является надежным средством подтверждения полномочий источника данных. Кроме того, все данные передаются по сети в открытом текстовом виде, поэтому их легко можно перехватить, например, с помощью утилиты `tcpdump` (или другого сканера пакетов):

```
$ zabbix_sender -v -z 10.10.2.9 -s alpha -k sniff.me -o "clear text data"

$ tcpdump -s0 -nn -q -A port 10051
00:58:39.263666 IP 10.10.2.11.43654 > 10.10.2.9.10051: tcp 113
E....l@.@.P..........'C..."^......V.......
.Gp|.Gp|{
      "request":"sender data",
      "data":[
            {
              "host":"alpha",
              "key":"sniff.me",
              " value":"clear text data"}]}
```

Конечно, простые данные мониторинга или конфигурации, кажется, не требуют особой защиты, но их подделка может привести к ошибочным выводам.

Несмотря на отсутствие стандартных мер противостояния этой проблеме, существует несколько возможных решений, от простых, но не безопасных, до сложных и очень надежных. Имейте в виду, что эта книга не посвящена проблемам сетевой безопасности, поэтому вы не найдете здесь подробных инструкций по выбору и реализации собственного VPN-решения. Мы лишь коротко рассмотрим методы повышения безопасности взаимодействий между компонентами Zabbix, чтобы вы могли получить практическое понимание проблемы и смогли принять обоснованное решение.

Отказ от конфигурации сети

Если по каким-то причинам никакие другие решения невозможны, тогда просто указывайте исходящий IP-адрес для каждой ловушки (траппера) Zabbix, чтобы усложнить возможность подделки данных мониторинга с использованием утилиты `zabbix_sender`. Используйте макрос `{HOST.CONN}` в шаблоне, чтобы каждый хост автоматически применял собственный IP-адрес, как показано на рис. 2.10.

Не менее важно отключить возможность выполнения удаленных команд на агентах. То есть параметру `EnableRemoteCommands` в файле `zabbix_agentd.conf` необходимо присвоить значение `0`. Это приведет к некоторой потере удобства, но если нет возможности защитить и аутентифицировать взаимодействия между агентом и сервером, угроза безопасности слишком велика, даже для простого рассмотрения ее допустимости.

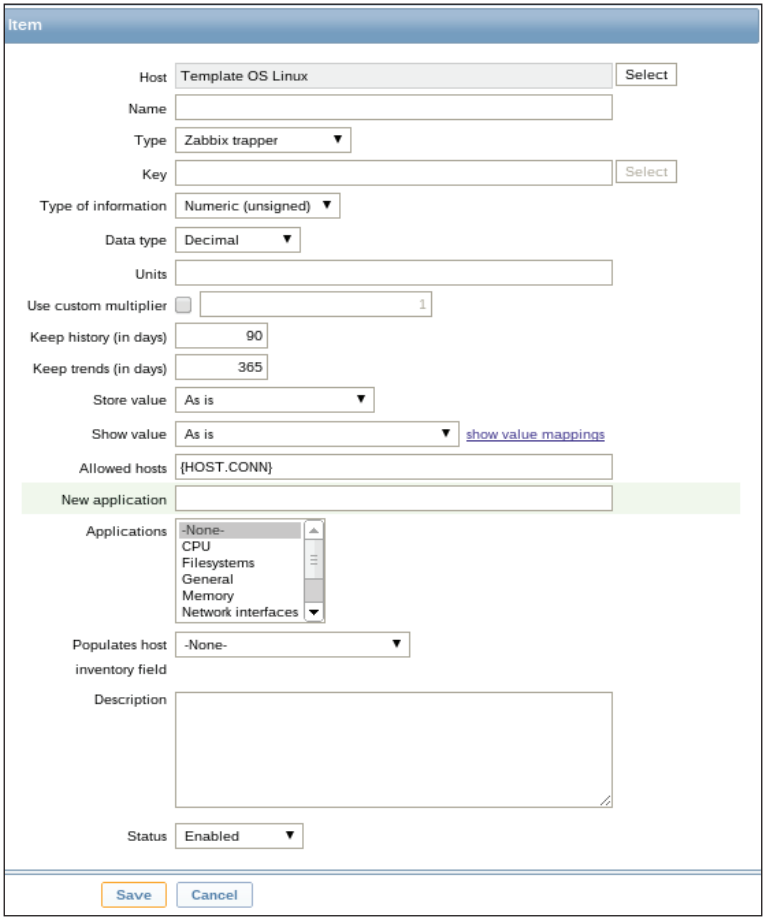


Рис. 2.10 ❖ Используйте макрос `{HOST.CONN}`

Изолирование сети

Во многих окружениях имеются сети управления, изолированные от эксплуатационных сетей посредством немаршрутизируемых сетевых адресов и виртуальных сетей. Сетевые коммутаторы, маршрутизаторы и брандмауэры, обслуживающие обычный сетевой трафик, достижимы и могут управляться только с адресов, принадлежащих управляющей сети. Это делает невозможным доступ к ним с произвольного рабочего места, но гарантирует, что любой недостаток в системе безопасности (представьте, например, сетевое устройство, имеющее дефектную реализацию SSL, которая исключает возможность ее использования, не поддерживает SNMP v3 или открывает неограниченную возможность использования Telnet) останется в отдельной, труднодостижимой сети. Вы можете включить в такую изолированную сеть все прокси-серверы и выполнять в ней все взаимодей-

ствия вида ведущий/ведомый. Изолирование трафика просто осложнит возможность перехвата данных мониторинга, но этой мерой не стоит пренебрегать, даже если вы собираетесь в дальнейшем шифровать трафик с применением методов, описанных далее.

С другой стороны, такой подход нежелательно применять к узлам или прокси-серверам, находящимся в защищенных сегментах сети (DMZ). Гораздо опаснее прокладывать маршруты через сеть управления в обход брандмауэра, чем передавать данные мониторинга через него. Конечно, это не относится к случаям, когда сеть управления контролируется брандмауэром, но я настоятельно рекомендую убедиться, что такой брандмауэр действительно имеется, прежде чем настраивать передачу данных мониторинга.

Простые туннели

До настоящего момента мы не предпринимали никаких мер по защите и шифрованию данных мониторинга в инфраструктуре Zabbix. Самый простой и непосредственный способ защитить данные – создать шифрованный туннель.

Протокол SSH

К счастью, протокол **SSH (Secure Shell** – защищенная оболочка) имеет встроенную возможность туннелирования, то есть у вас уже есть все необходимые инструменты для шифрования трафика.

Чтобы зашифровать трафик, движущийся от активного прокси-сервера к центральному серверу, просто откройте консоль на прокси-сервере и выполните команду:

```
$ ssh -N -f user@zabbix.server -L 10053:localhost:10051
```

Ключ `-N` сообщает этой команде, что вы запрещаете клиенту SSH выполнять произвольные команды и разрешаете только передавать трафик; ключ `-f` требует от клиента SSH перейти в фоновый режим (чтобы не держать окно терминала открытым или заставлять сценарий выполнять бесконечный цикл); `user@zabbix.server` – это имя пользователя и имя хоста (или IP-адрес) сервера Zabbix, и ключ `-L port:remote-server:port` определяет параметры туннеля. Первое значение (`port`) – это номер порта, используемый локальными приложениями, а следующая пара значений (`remote-server:port`) определяет имя хоста и TCP-порт сервера SSH с другого конца туннеля.

Теперь нужно настроить параметры `Server` и `ServerPort` в файле `zabbix_proxy.conf`, присвоив им значения `localhost` и `10053`, соответственно.

С этого момента прокси-сервер автоматически будет посылать данные в порт `10053`, а клиент SSH, прослушивающий его, будет переправлять весь трафик серверу Zabbix по протоколу SSH. Сервер SSH, в свою очередь, передаст полученные данные в локальный порт `10051`, который прослушивает демон Zabbix. Таким способом, даже при том, что компоненты Zabbix не поддерживают шифрования данных, вы сможете обеспечить целостность и конфиденциальность сообщений, курсирую-

щих между прокси и центральным сервером. В сети этот трафик будет выглядеть как обычный шифрованный поток данных, передаваемый в TCP-порт 22.

Чтобы проложить туннель между сервером Zabbix и пассивным прокси-сервером, на стороне прокси нужно настроить сервер SSH (он уже должен быть настроен в процессе подготовки машины к администрированию) и выполнить команду, представленную выше, на стороне сервера Zabbix, не забыв подставить в нее допустимый IP-адрес и имя пользователя для прокси Zabbix. Измените IP-адрес прокси и порт подключения в веб-интерфейсе, и на этом настройку можно считать законченной.

Чтобы подключить узлы Zabbix, нужно настроить два таких туннеля, один от ведущего к ведомому, а второй – от ведомого к ведущему.

На стороне ведущего выполните следующую команду:

```
$ ssh -N -f user@zabbix.child -L 10053:localhost:10051
```

На стороне ведомого:

```
$ ssh -N -f user@zabbix.master -L 10053:localhost:10051
```

Учитывая особую важность туннеля SSH, хорошей практикой считается настройка отправки SSH-клиентом контрольных пакетов, поддерживающих туннель открытым, например:

```
ssh -o ServerAliveInterval=60 -N -f user@zabbix.[child|master] -L
10053:localhost:10051
```

В совете выше показано, как настроить отправку контрольных пакетов; значение `ServerAliveInterval` выражается в секундах и представляет интервал между последовательными пакетами, обеспечивающими поддержку сеанса в открытом состоянии. Кроме того, нелишним будет организовать наблюдение за каналом и при обнаружении проблем закрывать его и открывать вновь.

Один из способов организовать такой мониторинг за туннелем SSH – добавить параметр

```
ExitOnForwatdFailure=yes
```

в командную строку. После этого нам останется только проверять наличие процесса клиента, потому что с этим параметром он будет завершаться при появлении ошибок.

Программа stunnel

Аналогичную функциональность можно получить от программы `stunnel`. Главное преимущество `stunnel` перед клиентом SSH заключается в возможности сохранить настройки туннеля в конфигурационном файле, тогда как при использовании клиента SSH приходится создавать сценарии с командой из предыдущего раздела, если необходимо, чтобы туннели восстанавливались после перезагрузки системы.

После установки программы и создания копий сертификатов SSL, необходимых ей, достаточно определить в файле `/etc/stunnel/stunnel.conf` все порты, подле-

жащие перенаправлению. Рассмотрим для примера простой сценарий с сервером Zabbix, который принимает данные от активного прокси-сервера и обменивается данными с другим узлом.

Для начала необходимо установить `stunnel` и сертификаты SSL на все три машины. Затем, на стороне сервера Zabbix, в файл `stunnel.conf` нужно добавить следующие строки:

```
[proxy]
accept = 10055
connect = 10051

[node - send]
accept = localhost:10057
connect = node.server:10057

[node - receive]
accept = 10059
connect = 10051
```

На стороне прокси-сервера:

```
[server]
accept = localhost:10055
connect = zabbix.server:10055
```

И на стороне узла:

```
[node - send]
accept = localhost:10059
connect = node.server:10059

[node - receive]
accept = 10057
connect = 10051
```

При этом важно не забыть отредактировать имя хоста и номер порта в соответствующих конфигурационных файлах центрального сервера и прокси, а также в веб-интерфейсе.

Как видите, чем больше туннелей требуется настроить, тем больше портов приходится задействовать. Большое количество прокси-серверов и узлов или добавление шифрования трафика от агентов усложняет выбор и настройку портов. Решение на основе `stunnel` хорошо подходит, когда требуется шифровать трафик между небольшим количеством хостов, но если требуется гарантировать шифрование всего трафика в системе мониторинга, необходимо использовать более полноценную реализацию VPN.

Использование полноценной VPN

Мы не будем обсуждать здесь достоинства разных реализаций VPN, но если в вашей сети уже используется какое-то решение VPN, подумайте о возможности подключения всей инфраструктуры мониторинга Zabbix к шифрованному кана-

лу. Если вы не желаете, чтобы весь мир видел ваши данные, такое решение становится практически обязательным для двух узлов или сервера и прокси, находящихся в разных географических регионах и связанных посредством Интернета. В подобных случаях сеть VPN обычно уже настроена либо с помощью SSL, либо с помощью одного из решений IPsec. Если у вас нет такой сети, защита Zabbix-трафика может служить отличным поводом создать ее.

Описанные выше решения помогут защитить трафик и обеспечить простую аутентификацию хостов, но имейте в виду, что пока Zabbix не поддерживает защищенных протоколов на уровне приложения, туннелирование и шифрование помогут лишь защитить целостность данных мониторинга. Любой пользователь, имеющий доступ к компоненту Zabbix (серверу, прокси или агенту), сможет послать поддельные данные через шифрованный канал, и у вас не будет никакой возможности заметить подделку. Поэтому, в дополнение к защите коммуникационных каналов, также необходимо гарантировать хорошую защищенность на уровне хоста. Начиная с версии Zabbix 3.0 будет поддерживаться шифрование TLS между сервером, агентом и прокси. Но эта возможность появится только в версии Zabbix 3.0. А до тех пор нам придется продолжать использовать альтернативные методы, описанные в этой главе.

В заключение

В данной главе мы видели, как расширить простую автономную инфраструктуру Zabbix до полноценного распределенного решения мониторинга. К настоящему времени вы должны понимать, как работают прокси-серверы Zabbix, как они переправляют данные мониторинга, в чем их достоинства и недостатки и какие требования к оборудованию и обслуживанию они предъявляют.

Вы также узнали, когда и как использовать активные или пассивные прокси-серверы, в каких случаях следует использовать более надежные базы данных, такие как MySQL, и, что еще более важно, как объединить эти возможности в уникальном решении для вашего окружения.

В заключение вы получили четкое представление, как оценить проблемы безопасности данных мониторинга и какие меры можно предпринять, чтобы снизить риски для инфраструктуры Zabbix.

В следующей главе мы завершим обзор вопросов развертывания инфраструктуры Zabbix в большом окружении и поговорим о высокой доступности на трех уровнях: базы данных, сервера мониторинга и веб-интерфейса.

Глава 3

Высокая доступность и отказоустойчивость

Теперь, получив хорошее представление обо всех компонентах инфраструктуры Zabbix, можно заняться обеспечением высокой доступности. В большом окружении, особенно если требуется обеспечить бесперебойную работу всех серверов, жизненно важно иметь надежную инфраструктуру Zabbix. Система мониторинга и инфраструктура Zabbix должны сохранять работоспособность при любых авариях и гарантировать непрерывную работу окружения.

Высокая доступность – одно из решений, гарантирующих непрерывность функционирования и реализующих восстановление в аварийных ситуациях; мы не можем пройти мимо этих решений в данной книге.

Эта глава начинается с определения понятия высокой доступности и затем описывает, как ее реализовать.

В частности, в этой главе рассматривается трехуровневая конфигурация, описывавшаяся в предыдущих главах:

- веб-интерфейс Zabbix;
- сервер Zabbix;
- базы данных.

Далее будет описано, как настроить каждый из компонентов для работы в режиме высокой доступности. Все процедуры, представленные в этой главе, были реализованы и опробованы на практике.

Эта глава охватывает следующие темы:

- что такое высокая доступность, отказоустойчивость и уровень обслуживания;
- глубокий анализ всех компонентов (сервер Zabbix, веб-сервер и сервер СУБД) инфраструктуры и как обеспечивается их высокая доступность;
- настройка высокой доступности инфраструктуры мониторинга.

Высокая доступность

Высокая доступность (high availability) – это подход к проектированию архитектуры и реализации соответствующих служб, цель которого – гарантировать

надежность обслуживания. Доступность напрямую связана со временем непрерывной работы и удобством использования службы. Это означает необходимость уменьшения периодов простоя до уровня, соответствующего соглашению об уровне обслуживания.

Периоды простоя можно разделить на две категории:

- запланированные простои;
- незапланированные, или неожиданные, простои.

Запланированные простои можно, в свою очередь, разделить на:

- прием исправлений в систему;
- добавление, модернизацию и замену оборудования;
- обслуживание программного обеспечения;
- любые другие плановые работы, связанные с обслуживанием инфраструктуры.

К сожалению, все эти простои прерывают обслуживание клиентов, но их проведение можно запланировать в согласованные периоды времени.

Незапланированные простои обычно возникают в случае отказов, которые могут быть вызваны следующими причинами:

- человеческая ошибка;
- выход оборудования из строя;
- программная ошибка;
- физические события.

Незапланированные простои также возникают из-за отключения электроэнергии и перегрева, и все они происходят совершенно неожиданно. Что такое выход из строя оборудования или программная ошибка, всем понятно, а вот под физическими событиями здесь подразумеваются любые внешние события, влекущие выход из строя нашей инфраструктуры. Примерами таких физических событий могут служить удар молнии или наводнение, влекущие за собой выход из строя электрических сетей и, соответственно, прекращение работы нашей инфраструктуры. Доступность службы оценивается с точки зрения пользователя. Например, выполняя мониторинг веб-приложения, мы должны подходить к поставленной задаче с позиции пользователя. То есть если все серверы работают, но брандмауэр не пропускает сетевых пакетов к службе, такую службу нельзя признать доступной.

Уровни обслуживания

Доступность непосредственно связана с уровнем обслуживания и обычно определяется в процентах. Этот процент вычисляется как отношение времени работы к заданному периоду. Доступность, которую вы можете гарантировать, и является вашим уровнем обслуживания. Таблица 3.1 точно отражает, что под этим подразумевается, перечисляя максимально допустимое время простоя для наиболее часто используемых процентов доступности.

Таблица 3.1 ❖ Время простоя для наиболее часто используемых процентов доступности

Доступность	Максимальное время простоя в год	Максимальное время простоя в месяц	Максимальное время простоя в неделю
90% (*одна девятка*)	36,5 дня	72 часа	16,8 часа
95%	18,25 дня	36 часов	8,4 часа
99% (*две девятки*)	3,65 дня	7,20 часа	1,68 часа
99,5%	1,83 дня	3,60 часа	50,4 минуты
99,9% (*три девятки*)	8,76 часа	43,8 минуты	10,1 минуты
99,99% (*четыре девятки*)	52,53 минуты	4,32 минуты	1,01 минуты
99,999% (*пять девяток*)	5,26 минуты	25,9 секунды	6,05 секунды
99,9999% (*шесть девяток*)	31,5 секунды	2,59 секунды	0,605 секунды
99,99999% (*семь девяток*)	3,15 секунды	0,259 секунды	0,0605 секунды

Время непрерывной работы (uptime) не является синонимом доступности. Система может продолжать работать, но оставаться недоступной; например, в случае выхода из строя компьютерной сети служба будет недоступна, но все системы будут продолжать работать.

Доступность должна вычисляться как сквозная оценка, вклад в которую вносят все компоненты, необходимые службе для работы. Следующее предложение может показаться парадоксальным; чем больше оборудования вы добавляете и чем больше точек отказа создаете, тем сложнее реализовать эффективное решение. Немаловажное значение имеют простота модернизации высокодоступной системы и ее обслуживание. По-настоящему высокодоступные системы не требуют вмешательства человека; например, если вы декларировали уровень обслуживания в *пять девяток*, у человека (вашего системного администратора) будет только одна секунда в день, чтобы устранить простой, поэтому подобные проблемы система должна решать автоматически. С другой стороны, если вы декларировали уровень обслуживания в *две девятки*, допускается продолжительность простоя до 15 минут в день, и в данном случае решением проблем вполне может заняться и человек, но, к сожалению, в наши дни такой уровень обслуживания не сможет выиграть в конкурентной борьбе. Не менее важную роль играет еще один фактор – среднее время восстановления.

Среднее время восстановления (Mean Time To Recovery, MTTR) – это время, необходимое устройству для восстановления после отказа.

В первую очередь необходимо сохранить архитектуру максимально простой и свести к минимуму число составляющих ее компонентов. Чем проще архитектура, тем меньше усилий она потребует на сопровождение, администрирование и мониторинг. Все, что необходимо для организации высокой доступности, – избежать создания одиночных точек отказа и сохранить архитектуру максимально простой. По этой причине решение, представленное здесь, понятно, проверено на практике, легко реализуется и не вызывает сложностей в сопровождении.

Сложность – первый враг высокой доступности.

К сожалению, высокодоступная инфраструктура не способна обеспечить максимальную производительность. Это объясняется накладными расходами, связанными с копированием данных на два сервера. И высокодоступная инфраструктура не способна обеспечить максимальную пропускную способность. Кроме того, существуют реализации, в которых резервный сервер действует как сервер только для чтения, чтобы уменьшить нагрузку на основной узел.

Высокодоступная архитектура не способна обеспечить максимальную производительность или пропускную способность.

Некоторые вопросы высокой доступности

Все архитектуры высокой доступности имеют общие проблемы, которые приходится решать в процессе реализации:

- как обрабатывать соединения;
- как управлять отказами;
- как обеспечить репликацию и совместное использование хранилищ.

Для каждой из проблем, перечисленных выше, давно придуманы надежные решения. Рассмотрим их подробнее.

- **Как обрабатывать соединения?**
  Один из возможных ответов на этот вопрос – применять **виртуальные IP-адреса (Virtual IP, VIP)**. Практически каждый программный компонент инфраструктуры должен взаимодействовать или связывать разные логические уровни, и эти компоненты часто развертываются на разных серверах, чтобы обеспечить равномерное распределение нагрузки. Большая часть взаимодействий осуществляется по протоколу TCP/IP, который готов протянуть нам руку помощи.
  Существует возможность определить виртуальный IP-адрес, присвоить его активным серверам и настроить все программное обеспечение на использование этого адреса. Тогда, в случае отказа, IP-адрес будет подхвачен следующим сервером, и все клиенты продолжат работу, как ни в чем не бывало. Конечно, это решение не может гарантировать полного отсутствия простоев, но способно ограничить их очень короткими периодами времени. Администратору не нужно предпринимать никаких действий для перенастройки в случае сбоя.
- **Как управлять отказами?**
  Ответ на этот вопрос: использовать диспетчера ресурсов. Вам потребуется придумать некоторый способ максимально быстрой передачи управления резервному серверу в случае отказа. Для достижения минимального времени простоя необходимо, чтобы такая передача происходила автоматически, а причина отказа должна быть найдена как можно быстрее.

- **Как обеспечить репликацию и совместное использование хранилищ?** Решение этой последней проблемы можно реализовать разными методами и с привлечением разных технологий. Можно использовать разделяемый диск, дублировать **логический номер устройства** (Logical Unit Number, LUN) между двумя хранилищами или дублировать устройство с программным обеспечением. К сожалению, дублирование логического номера устройства между двумя хранилищами обходится недешево. Соответствующее программное обеспечение должно выполняться как можно ближе к ядру и работать на как можно более низком уровне, чтобы обеспечить максимальную прозрачность перехода и тем самым упростить обслуживание.

Автоматизация аварийного переключения с применением диспетчера ресурсов

Архитектура с высокой доступностью должна содержать компонент, автоматизирующий аварийное переключение. Как уже говорилось выше, таким компонентом является диспетчер ресурсов.

В качестве примера диспетчера ресурсов, пригодного к промышленной эксплуатации, можно привести **Pacemaker**. Pacemaker – это открытый диспетчер ресурсов, предназначенный для использования в небольших и крупных кластерах. Pacemaker можно получить по адресу: http://clusterlabs.org/.

Этот диспетчер ресурсов обладает следующими интересными возможностями, по-настоящему полезными для кластеров:

- определение и решение проблем на уровне приложений;
- поддержка избыточных конфигураций;
- поддержка приложений на множестве узлов;
- поддержка запуска/остановки приложений в определенном порядке.

Практически Pacemaker может заменить администратора и действовать намного быстрее. Pacemaker выполняет те же действия, которые обычно выполняются администратором Unix в случае отказа узла. Он проверяет доступность службы и при необходимости производит аварийное переключение всех настроенных служб на другой узел; к тому же делает эту работу максимально быстро. Автоматизация переключения дает нам время на анализ причин сбоя и обеспечивает доступность служб.

Существуют также другие решения управления кластерами. Популярной альтернативой, например, является Red Hat Cluster Suite. Мы не будем рассматривать ее в этой книге, потому что она тесно связана с дистрибутивом Red Hat, поскольку то разрабатывалась специально для него.

Репликация файловой системы с помощью DRBD

DRBD (Distributed Replicated Block Device – распределенное копируемое блочное устройство) обладает особенностями, которые являются важнейшими достоинствами этого решения:

- это модуль ядра;
- действует совершенно прозрачно для СУБД;
- обеспечивает синхронизацию в масштабе реального времени;
- синхронизирует запись на оба узла;
- автоматически выполняет повторную синхронизацию;
- действует практически как сетевой RAID 1.

Основная функциональность DRBD реализована на уровне ядра; в частности, DRBD – это драйвер виртуального блочного устройства, действующий в самом низу стека системы ввода/вывода.

DRBD можно рассматривать как эквивалент сетевого дискового массива RAID 1, расположенный ниже файловой системы.

Это означает, что синхронизация DRBD происходит одновременно с файловой системой. Худшим и самым сложным случаем является репликация базы данных. В этой ситуации каждое подтверждение транзакции должно выполняться на двух узлах, и все подтвержденные транзакции записываются на оба узла; DRBD полностью поддерживает этот сценарий.

Но что случится, когда один из узлов окажется недоступен? Все просто: DRBD продолжит работу как деградированный RAID 1. Это мощная особенность, потому что в случае отказа вам ничего не придется делать. Как только узел вновь появится в сети, DRBD автоматически выполнит синхронизацию.

Реализация высокой доступности для веб-сервера

Теперь, когда вы познакомились со всеми компонентами, участвующими в игре, можно приступить к реализации высокой доступности для веб-сервера. В предлагаемой конфигурации предусматривается использование веб-сервера Apache с присвоенным ему виртуальным IP-адресом, на двух узлах, входящих в состав кластера, который управляется диспетчером Corosync/Pacemaker.

В обеспечении высокой доступности веб-интерфейса Zabbix нет ничего сложного, потому что веб-приложение, выполняющееся на веб-сервере, не генерирует никаких данных или файлов. Это позволяет развернуть два узла на двух разных серверах – возможно, географически удаленных друг от друга – и реализовать отказоустойчивую конфигурацию с высокой доступностью. В этой конфигурации нет необходимости предусматривать репликацию файловых систем между двумя узлами, так как информационное наполнение имеет *статический* характер, в том смысле, что не изменяется (за исключением случаев модернизации системы). Единственный дополнительный компонент, который нам понадобится, – это диспетчер ресурсов. Он будет определять сбой первичного узла и координировать аварийное переключение на вторичный узел. В роли диспетчера ресурсов будет использоваться Pacemaker/Corosync.

Установка выполняется в следующем порядке:

1. Установка сервера HTTPD (Apache) на оба узла.
2. Установка Pacemaker.

3. Развертывание веб-интерфейса Zabbix на обоих узлах.
4. Настройка виртуального IP-адреса в Apache.
5. Настройка Corosync/Pacemaker.
6. Настройка доступа к СУБД в веб-интерфейсе Zabbix (по виртуальному IP-адресу PostgreSQL).

Описываемая инфраструктура изображена на рис. 3.1.

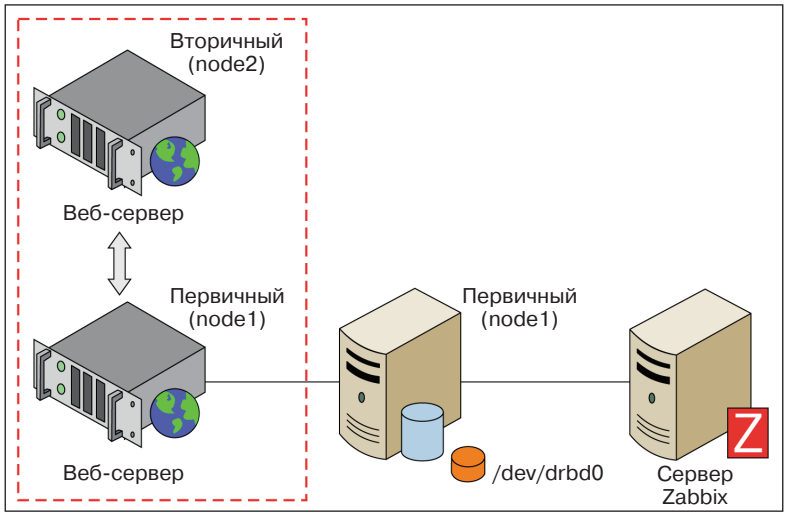


Рис. 3.1 ❖ Инфраструктура мониторинга с двумя веб-серверами

Настройка HTTPD

Pacemaker – мощный диспетчер ресурсов, широко используемый и обладающий множеством интересных возможностей. Чтобы установить и настроить Pacemaker, необходимо:

- установить Corosync;
- установить Pacemaker;
- настроить и запустить Corosync.

Уделим немного внимания этой части архитектуры. Corosync – это программный слой, реализующий службу обмена сообщениями между серверами, объединенными в кластер.

Corosync поддерживает кластеры любого размера и позволяет использовать разные конфигурации, обеспечивающие устойчивость к отказам, такие как активный–активный, активный–пассивный и N+1. Одной из задач Corosync являются проверка наличия процесса Pacemaker и запуск всех необходимых процессов.

Устанавливается пакет следующей командой:

```
$ yum install pacemaker corosync
```

Диспетчер yum автоматически определит все зависимости. После установки можно приступать к настройке Corosync. В первую очередь необходимо скопировать шаблон конфигурационного файла:

```
$ cp /etc/corosync/corosync.conf.example /etc/corosync/corosync.conf
```

Чтобы настроить Corosync, выберите неиспользуемый групповой адрес и порт:

```
$ export MULTICAST_PORT=4000
$ export MULTICAST_ADDRESS=226.94.1.1
$ export BIND_NET_ADDRESS=`ip addr | grep "inet " |grep brd |tail -n1 | awk '{print $4}' |
sed s/255/0/`

$ sed -i.bak "s/.*mcastaddr:.*/mcastaddr:\ $MULTICAST_ADDRESS/g" /etc/corosync/corosync.
conf
$ sed -i.bak "s/.*mcastport:.*/mcastport:\ $MULTICAST_PORT/g" /etc/corosync/corosync.conf
$ sed -i.bak "s/.*bindnetaddr:.*/bindnetaddr:\ $BIND_NET_ADDRSS/g" /etc/corosync/corosync.
conf
```

Не забудьте настроить правило для брандмауэра, пропускающее групповой трафик в порт 4000, выполнив следующую команду от имени root:

```
iptables -I INPUT -p udp -m state --state NEW -m multiport --dports 4000 -j ACCEPT
```

После этого выполните команду:

```
service iptables save
```

Теперь необходимо добавить в Corosync службу Pacemaker и создать файл /etc/corosync/service.d/pcmk со следующим содержимым:

```
service {
# Загрузить диспетчера ресурсов Pacemaker
name: pacemaker
ver: 1
}
```

Затем следует скопировать подготовленные файлы на второй узел (node2):

```
/etc/corosync/corosync.conf
/etc/corosync/service.d/pcmk
```

После этого можно запустить Corosync и Pacemaker на обоих узлах:

```
$ /etc/init.d/corosync start
$ /etc/init.d/pacemaker start
```

Проверьте состояние кластера следующей командой:

```
$ crm_mon
```

и конфигурацию – командой:

```
$ crm configure show
```

Pacemaker и механизм STONITH

Механизм **STONITH (Shoot The Other Node In The Head)**[1] может оказаться слабым звеном в этой конфигурации и привести к неправильному распределению полномочий между узлами, особенно если серверы удалены географически и имеется масса факторов, способных нарушить нормальное взаимодействие между ними. Такие нарушения возможны, например, когда каждый узел полагает, что другой потерпел аварию и теперь он является первичным. В этом случае, когда вторичный узел перезагрузится, он «выстрелит» в первого, и цикл повторится с точностью до наоборот. Эта проблема известна как клинч STONITH.

Существуют три причины, которые могут вынудить один узел «выстрелить» в другой:

- узлы работоспособны, но нарушено взаимодействие между ними;
- узел потерпел аварию;
- ресурс высокой доступности не смог остановиться.

Первую причину можно ликвидировать, наладив избыточные линии связи и организовав надежную групповую рассылку. Но для этого требуется реорганизовать всю сетевую инфраструктуру, и если вы приобретаете сетевые услуги у стороннего провайдера, вы не сможете положиться на надежность этих услуг и корректное управление групповыми рассылками. Вторая причина очевидна, и маловероятно, что она приведет к клинчу STONITH.

Третью причину понять сложнее, поэтому поясню ее на примере. Обычно ресурс высокой доступности запускается на узле. Если он запустился, начинается его непрерывный мониторинг. Если ресурс не смог запуститься, он будет остановлен и перезапущен на текущем или на вторичном узле. Если ресурс должен быть остановлен и остановка прошла успешно, он будет перезапущен на другом узле. Но если остановка не произошла, механизм STONITH оградит этот ресурс, посчитав такое развитие событий наиболее безопасным.

Если ресурс высокой доступности не может быть остановлен и механизм STONITH оградил узел, остается только самое худшее – остановить весь узел. Это может вызвать повреждение данных на узле, особенно при наличии незавершенных транзакций, и таких действий желательно избегать. Менее опасно, когда ресурсом высокой доступности является, например, сервер HTTPD, который просто возвращает веб-страницы (не выполняя никаких транзакций); но и в этом случае существует риск повреждения данных.

Существуют разные способы избежать клинча STONITH, но нам требуется, чтобы реализация оставалась максимально простой для управления и обслуживания, поэтому оставим предлагаемую архитектуру без внедрения механизма STONITH.

Pacemaker распространяется с включенной поддержкой механизма STONITH, который фактически не нужен в конфигурации с двумя узлами.

---

[1] Дословно переводится как «выстрел в голову другому узлу», но в данном случае имеется в виду возможность остановки/запуска/перезагрузки узла. – *Прим. перев.*

Чтобы отключить STONITH, выполните следующую команду:

```
$ crm configure property stonith-enabled="false"
```

Pacemaker – так ли необходим кворум?

Под кворумом в данном случае понимается идея голосования, когда каждый узел может проголосовать за выполнение того или иного действия. По аналогии с принципами демократии, когда большинство определяет решение, которое должно быть реализовано. Например, если имеется кластер с тремя (или более) узлами и один из узлов потерпел отказ, большинство может решить оградить отказавший узел.

В настройках кворума можно также определить, что делать в отсутствие кворума:

- **игнорировать**: ничего не делать в отсутствие кворума;
- **остановить** (по умолчанию): остановить все ресурсы на отказавшем узле кластера;
- **заморозить**: оставить выполняться все действующие ресурсы, но не запускать остановившиеся;
- **уничтожить**: оградить все узлы из раздела, где произошел отказ.

Кворум имеет смысл настраивать в кластерах с тремя и более узлами. По умолчанию в большинстве конфигураций кворум включен, но его нельзя использовать в кластерах с двумя узлами, потому что в этой конфигурации получить большинство голосов и принять решение невозможно.

Выполните следующую команду, чтобы применить правило `ignore` (игнорировать):

```
$ crm configure property no-quorum-policy=ignore
```

Pacemaker – идея закрепления ресурсов

Часто бывает желательно предотвратить перемещение действующих ресурсов между узлами кластера. Для перемещения ресурса всегда требуется некоторое время, что может быть недопустимо для критически важных служб (таких как СУБД), особенно если ресурс действует правильно. Для решения этой проблемы в Pacemaker имеется параметр, выражающий степень связи ресурса с узлом, где он действует в настоящий момент. Эта идея получила название **stickiness** (закрепление или прилипание). Всякий простой имеет свою цену, и простой, необходимый для переноса ресурса с одного узла на другой, не всегда укладывается в небольшой допустимый период.

Pacemaker не вычисляет стоимости переноса ресурсов и стремится достигнуть их оптимального размещения.

В кластере с двумя узлами очень важно определить степень связи ресурса с узлом; это упростит обслуживание инфраструктуры в дальнейшем. Pacemaker не должен принимать решения о переносе, если рассматриваемый ресурс действует как обычно.

Обратите внимание, что оптимальное размещение ресурсов с точки зрения Pacemaker не всегда согласуется с нашими представлениями. Чтобы исключить перемещение ресурсов, для каждого из них можно определить свою степень связи с узлом:

```
$ crm configure property default-resource-stickiness="100"
```

Для выражения степени связи вместо числа можно использовать константу `INFINITY`. В этом случае ресурс будет продолжать действовать на выбранном узле, пока тот не остановится, и как только первичный узел восстановит работоспособность, все ресурсы со значением `INFINITY` немедленно вернутся на него:

```
$ crm configure property default-resources-tickiness=" INFINITY"
```

Pacemaker – настройка Apache/HTTPD

Диспетчер ресурсов Pacemaker должен иметь доступ к информации о состоянии сервера Apache. Для этого следует изменить содержимое файла `/etc/httpd/conf.d/httpd.conf`, как показано ниже:

```
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1 <МАСКА-ВАШЕЙ-СЕТИ>/24
</Location>
```

По соображениям безопасности, желательно запретить доступ к этой информации отовсюду и разрешить только для вашей локальной сети и локального интерфейса (`127.0.0.1`).

После добавления этих изменений нужно перезапустить Apache командой от имени root:

```
$ service httpd restart
```

Эта конфигурация рассчитана на два веб-сервера, которые для простоты мы назовем `www01` и `www02`. А также, чтобы сохранить пример максимально простым, будем считать, что веб-серверы имеют следующие адреса:

- `www01` (eth0: 192.168.1.50, eth1: 10.0.0.50);
- `www02` (eth0: 192.168.1.51, eth1: 10.0.0.51).

Теперь выполним настройку виртуального адреса, выполнив следующие команды:

```
$ crm configure
crm(live)configure# primitive vip ocf:heartbeat:IPaddr2 \
> params ip="10.0.0.100"
# обратите внимание, что 10.0.0.100 - это IP-адрес pacemaker
> nic="eth1" \
> cidr_netmask="24" \
> op start interval="0s" timeout="50s" \
```

```
> op monitor interval="5s" timeout="20s" \
> op stop interval="0s" timeout="50s"
crm(live)configure# show
# чтобы убедиться
node www01.domain.example.com
node www02.domain.example.com
primitive vip ocf:heartbeat:IPaddr2 \
    params ip="10.0.0.100" nic="eth1" cidr_netmask="24" \
    op start interval="0s" timeout="50s" \
    op monitor interval="5s" timeout="20s" \
    op stop interval="0s" timeout="50s"
property $id="cib-bootstrap-options" \
    dc-version="1.1.2-f059ec7cedada865805490b67ebf4a0b963bccfe" \
    cluster-infrastructure="openais" \
    expected-quorum-votes="2" \
    no-quorum-policy="ignore" \
    stonith-enabled="false"
rsc_defaults $id="rsc-options" \
    resource-stickiness="INFINITY" \
    migration-threshold="1"

crm(live)configure# commit
crm(live)configure# exit
```

Команда `commit` сохраняет и активирует конфигурацию. Теперь, чтобы убедиться, что все работает, как требуется, проверим настройки следующей командой:

```
$ crm_mon
```

Она должна вывести:

```
============
Last updated: Fri Jul 10 10:59:16 2015
Stack: openais
Current DC: www01.domain.example.com - partition WITHOUT quorum
Version: 1.1.2-f059ec7cedada865805490b67ebf4a0b963bccfe
2 Nodes configured, , unknown expected votes
1 Resources configured.
============

Online: [ www01.domain.example.com www02.domain.example.com ]

vip (ocf::heartbeat:IPaddr2): Started www01.domain.example.com
```

Чтобы убедиться, что виртуальный IP-адрес действует, просто проверим его командой `ping`:

```
$ ping 10.0.0.100
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
64 bytes from 10.0.0.100: icmp_seq=1 ttl=64 time=0.012 ms
64 bytes from 10.0.0.100: icmp_seq=2 ttl=64 time=0.011 ms
```

```
64 bytes from 10.0.0.100: icmp_seq=3 ttl=64 time=0.008 ms
64 bytes from 10.0.0.100: icmp_seq=4 ttl=64 time=0.021 ms
```

Итак, мы настроили и активировали виртуальный IP-адрес. Чтобы настроить Apache в кластере, нужно вернуться назад в настройки CRM и сообщить Corosync, что у нас появилась новая служба – демон HTTPD – и его следует сгруппировать с виртуальным IP-адресом. Эта группа будет называться "web server".

Эти настройки свяжут виртуальный IP-адрес и HTTPD, и оба будут запущены на одном узле. Настроим использование виртуального IP-адреса следующими командами:

```
$ crm configure
crm(live)configure# primitive httpd ocf:heartbeat:apache \
> params configfile="/etc/httpd/conf/httpd.conf" \
> port="80" \
> op start interval="0s" timeout="50s" \
> op monitor interval="5s" timeout="20s" \
> op stop interval="0s" timeout="50s"
crm(live)configure# group webserver vip httpd
crm(live)configure# commit
crm(live)configure# exit
```

Теперь проверим конфигурацию:

```
$ crm_mon
============
Last updated: Fri Jul 10 10:59:16 2015
Stack: openais
Current DC: www01.domain.example.com - partition WITHOUT quorum
Version: 1.1.2-f059ec7cedada865805490b67ebf4a0b963bccfe
2 Nodes configured, unknown expected votes
1 Resources configured.
============

Online: [ www01.domain.example.com www02.domain.example.com ]

Resource Group: webserver
vip (ocf::heartbeat:IPaddr2): Started www01.domain.example.com
httpd (ocf::heartbeat:apache): Started www01.domain.example.com
```

Обратите внимание: так как мы не используем кворум, команда `crm_mon` вывела: `partition WITHOUT quorum` и `unknown expected votes`.

Реализация высокой доступности для сервера Zabbix

Кластер высокой доступности для сервера Zabbix настраивается проще, чем для веб-сервера Apache или сервера баз данных. Процедура настройки автономного сервера и узла, являющегося частью кластера, остается прежней, как показано на рис. 3.2.

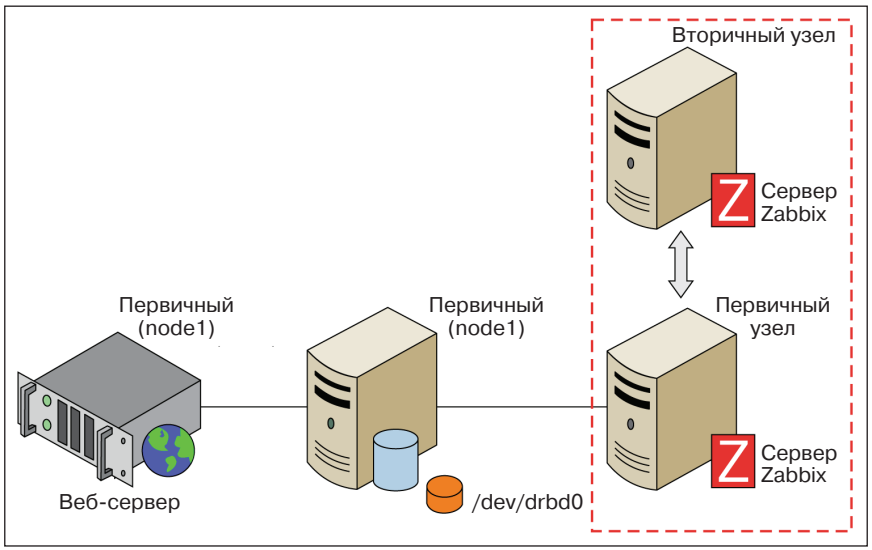


Рис. 3.2 ❖ Настройка автономного сервера и узла, являющегося частью кластера, остается прежней

После установки Corosync и Pacemaker на два узла (как описывалось в предыдущем разделе) на эти же узлы следует установить Zabbix. После этого нужно настроить систему Zabbix на прослушивание виртуального IP-адреса. Для этого требуется изменить оба параметра – `SourceIP` b `ListenIP` – в конфигурационном файле `zabbix_server.conf`:

```
SourceIP=10.10.1.9
ListenIP=10.10.1.9
```

Укажите в этих настройках виртуальный IP-адрес, который вы выбрали для своего кластера Zabbix.

Теперь можно продолжить и отключить STONITH:

```
$ crm configure property stonith-enabled="false"
```

Если кластер насчитывает всего два узла, необходимо так же отключить кворум; иначе кластер не сможет принять решение из-за отсутствия большинства:

```
$ crm configure property no-quorum-policy=ignore
```

И наконец, определите достаточно высокую степень связи, чтобы избежать автоматического перемещения службы между узлами взад и вперед до выхода из строя активного узла:

```
$ crm configure property default-resource-stickiness="100"
```

По аналогии с настройками Apache/HTTPD необходимо также определить примитив для виртуального IP-адреса:

```
$ crm configure primitive Zbxvip ocf:heartbeat:IPaddr2 \
params ip="10.10.1.9" iflabel="httpvip" \
op monitor interval="5"
```

Определите примитив для сервера Zabbix, выполнив следующую команду:

```
$ crm configure primitive Zabbix lsb::zabbix_server \
op monitor interval="5"
```

Так же как в предыдущем разделе, осталось лишь сгруппировать примитивы, связать с узлами и определить порядок запуска:

```
$ crm configure group Zbx_server Zbxvip Zabbix meta target-role="Started"
$ crm configure colocation Ip_Zabbix inf: Zbxvip Zabbix
$ crm configure order StartOrder inf: Zbxvip Zabbix
```

Как видите, чем проще компоненты, тем легче настроить их для работы в кластере под управлением Pacemaker. Но ситуация в корне меняется, когда дело доходит до настройки наиболее важной части: базы данных или любого другого хранилища данных.

Реализация высокой доступности для базы данных

Реализация высокой доступности для базы данных – очень непростая задача. Существует много способов ее решения с разной степенью сложности и с использованием различного программного обеспечения.

В этом разделе предлагается довольно избыточное решение, но имейте в виду, что это лишь одно из возможных решений, широко используемых в больших окружениях. Для реализации данного решения понадобятся два одинаковых сервера баз данных, установленных на двух компьютерах с одинаковыми операционными системами и одинаковым набором программного обеспечения. Так как серверы должны быть похожи как близнецы и тесно связаны друг с другом, очевидно, что они должны иметь один и тот же комплект программного обеспечения идентичных версий.

Поскольку будут работать два физически разных сервера, должна быть обеспечена репликация данных между ними; это предполагает необходимость соединения серверов выделенной сетью, обеспечивающей необходимую пропускную способность.

В этой конфигурации серверы могут находиться в одном или в разных вычислительных центрах, которые должны обеспечивать надежную, отказоустойчивую связь. Только в этом случае можно гарантировать высокую доступность.

Существуют разные способы репликации данных между двумя серверами:

- репликация файловых систем;
- общий отказоустойчивый диск;
- горячее/теплое резервирование с использованием PITR (Point-in-Time Recovery – непрерывное архивирование и восстановление на момент времени);

- репликация мастер–резерв на основе триггеров (Trigger-Based Master-Standby Replication);
- ПО для репликации на основе выражения (Statement-Based Replication Middleware);
- асинхронная репликация при наличии нескольких мастеров (Asynchronous Multimaster Replication);
- синхронная репликация с мастером (synchronous master replication).

Каждый из них имеет свои достоинства и недостатки. Из этих вариантов можно исключить все решения на основе триггеров, потому что они увеличивают нагрузку на ведущий узел. Кроме того, дополнительный уровень на пользовательском уровне может вносить искажения и неточности.

Среди перечисленных решений есть несколько, которые гарантируют малое, по-настоящему малое время на восстановление после сбоя и безопасны с точки зрения потери данных. Следующие решения гарантируют сохранность данных в случае аварии ведущего узла (мастера):

- общий отказоустойчивый диск;
- репликация файловых систем;
- ПО для репликации на основе выражения.

Решение на основе общедоступного отказоустойчивого диска подразумевает использование общей сети хранения данных (Storage Area Network, SAN). То есть если вы захотите поместить свой сервер в отдельную ферму в другом месте, такая система обойдется вам очень дорого.

Решение на **основе горячего/теплого резервирования с использованием PITR** требует большого объема свободного пространства для обработки и сохранения всех файлов журналов, сгенерированных транзакциями, в случае отказа узла. Эта конфигурация требует создания вторичной базы данных (идентичной основной на ведущем узле), которая работает в режиме горячего резерва и ждет, пока поступит транзакция. Как только транзакция будет получена, СУБД применяет ее ко вторичному узлу.

В этом случае, если вторичный узел выйдет из строя, необходимо предпринять дополнительные меры, потому что первичная база данных может произвести обширный архив неотправленных файлов журналов и вызвать остановку инфраструктуры. В больших окружениях обычно производится большое количество транзакций, и если проблема возникает в нерабочее время, конфигурация высокой доступности должна уметь обрабатывать ее.

Еще один способ – синхронная репликация PostgreSQL. В случае выхода из строя вторичного узла это решение требует перезагрузки, чтобы предотвратить зависание транзакций.

Решения на основе триггеров сложны и небезопасны, так как предполагают, что триггер может срабатывать с каждой операцией вставки и копировать эту вставку на вторичный узел, вводя дополнительные накладные расходы. Этот метод плохо поддерживает секционирование с наследованием. Кроме того, он не гарантирует сохранности данных в случае отказа мастера.

Инфраструктуры, включающие резервную базу данных, вводят в игру второго актера, то есть если база данных выйдет из строя или станет недоступной, это не должно вызвать зависание мастера. В настоящее время, при использовании PostgreSQL 9.1, вполне жизнеспособным является решение с применением синхронной репликации. К сожалению, такие решения добавляют определенные ограничения: подтверждение транзакции происходит только после передачи изменений в резервную базу, а факт передачи еще не означает, что произойдет копирование.

На практике это означает, что если вторичный узел выйдет из строя, первичная база данных повиснет, пока ведомый не примет транзакцию и не известит мастера об этом. В результате первичный узел может висеть неопределенное время, а это удваивает риск появления простоев.

Проблема на вторичном узле не должна влиять на работу первичного. Это удваивает риск появления простоев, что совершенно неприемлемо в контексте высокой доступности.

Кластеризация PostgreSQL

Кластер, представленный ниже, имеет простую организацию, спроектирован так, чтобы использовать как можно меньше компонентов, и предназначен для реализации высокой доступности.

Строение кластера показано на рис. 3.3. Он включает минимальное количество компонентов, прост в обслуживании, сопровождении и обновлении.

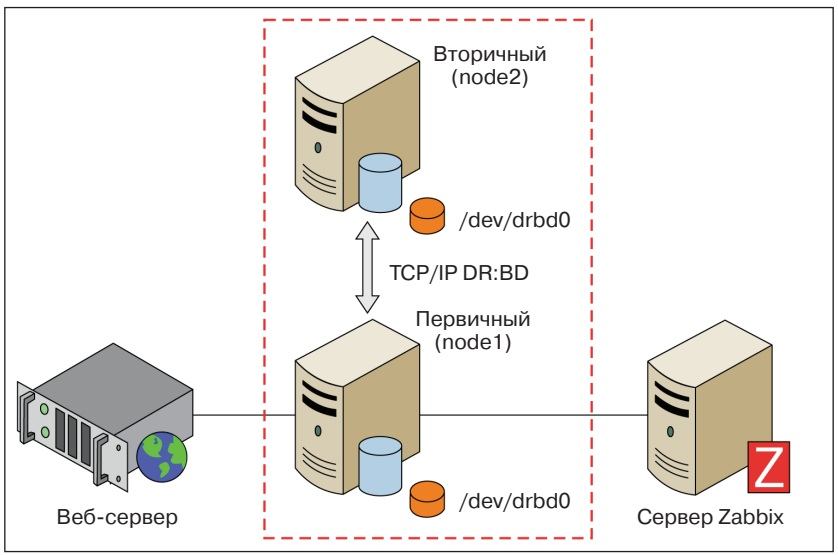


Рис. 3.3 ❖ Кластер СУБД PostgreSQL

Зеркалирование логического тома с помощью LVM и DRDB

LVM2 – это реализация механизма управления логическими томами (Logical Volume Manager, LVM), основанная на подсистеме виртуальных блочных устройств в Linux. Кроме названия, LVM2 не имеет ничего общего с LVM.

Ниже перечислены основные понятия LVM2 (см. также рис. 3.4).

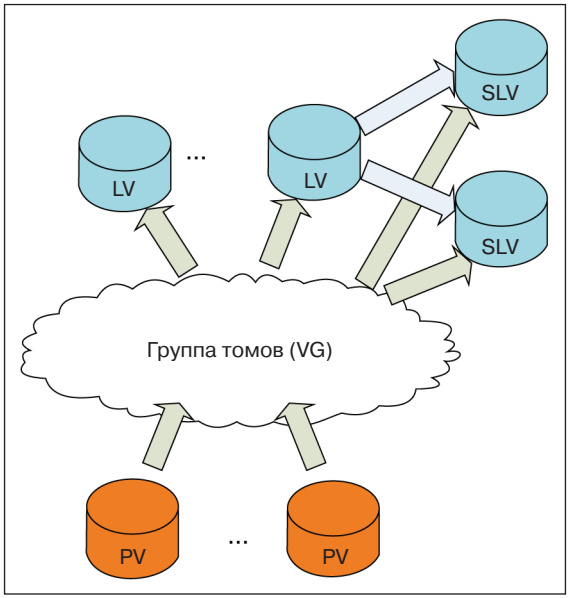


Рис. 3.4 ❖ Основные понятия LVM2

- **Физический том (Physical Volume, PV)**: физический раздел или система хранения, на которой построена система LVM.
- **Группа томов (Volume Group, VG)**: основная единица администрирования. Может включать один или несколько физических томов. Каждая группа томов имеет уникальное имя и может расширяться во время выполнения за счет добавления новых физических томов или увеличения размеров существующих.
- **Логический том (Logical Volume, LV)**: доступен в Linux как обычное блочное устройство, и его компоненты могут создаваться во время выполнения внутри доступных групп томов. Логические тома допускают динамическое изменение размеров и перемещение из одного физического тома в другой, если они входят в одну группу томов.
- **Мгновенная копия логического тома (Snapshot Logical Volume, SLV)**: это временная мгновенная копия логического тома. Даже если логический том имеет очень большой размер (порядка сотен гигабайт), для хранения мгновенной копии обычно требуется намного меньше пространства.

Сигнатура разделов 0x8E в Linux используется исключительно для маркировки разделов LVM. При этом разделы необязательно должны маркироваться этой сигнатурой. В действительности LVM распознает группы физических томов по сигнатуре, записанной во время инициализации PV.

Поскольку после создания логический том выглядит как обычное блочное устройство, его можно использовать совместно с DRBD.

Обязательные условия использования DRBD на LVM

Прежде чем настраивать DRBD на LVM, необходимо выполнить следующие предварительные условия:

- механизм LVM должен знать о существовании устройств DRBD;
- кэширование в LVM должно быть отключено;
- добавить в `initramfs` новые устройства ядра.

По умолчанию LVM сканирует все блочные устройства, присутствующие в `/dev`, пытаясь отыскать сигнатуры PV; соответственно, нужно настроить соответствующий фильтр в `/etc/lvm/lvm.conf`:

```
filter = ["a|sd.*|", "a|drbd.*|", "r|.*|"]
```

Этот фильтр принимает все диски SCSI и DRBD. Теперь нужно повторно просканировать все группы томов следующей командой:

```
# vgscan
```

Очень важно не забыть отключить кэширование в LVM, потому что диски DRBD исчезнут в случае отказа. Это нормальное явление при отказах, и если не отключить кэширование, может получиться так, что диск будет видим как доступный, когда его в действительности не существует.

Для отключения кэширования достаточно добавить следующую строку в `/etc/lvm/lvm.conf`:

```
write_cache_state = 0
```

После отключения кэширования на дисках еще может оставаться какая-то часть кэша, созданного перед этим. От нее нужно избавиться, удалив файл:

```
/etc/lvm/cache/.cache
```

Теперь нужно заново сгенерировать файлы карты устройств ядра командой:

```
# update-initramfs -u
```

После этого можно двигаться дальше и приступать к настройкам.

Создание устройства DRBD поверх раздела LVM

После отключения кэширования механизм LVM готов к дальнейшим настройкам. Сначала нужно создать физический том (PV). Чтобы инициализировать раздел SCSI как физический том, выполните следующую команду от имени root:

```
$ pvcreate /dev/sda1
  Physical volume "/dev/sda1" successfully created
$ pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```

Вывод сообщает, что тома были успешно инициализированы. Теперь можно создать низкоуровневую группу томов `vgpgdata`:

```
$ vgcreate vgpgdata /dev/sda1 /dev/sda2
  Volume group "vgpgdata" successfully created
```

Наконец, создадим том, точнее логический том, который будет использоваться как блочное устройство DRBD:

```
$ lvcreate --name rpgdata0 --size 10G local
  Logical volume "rpgdata0" created
```

Все эти шаги должны выполняться в том же порядке на обоих узлах. Теперь нужно установить DRBD на оба узла:

```
$ yum install drbd kmod-drbd
```

Для установки DRBD требуется подключить репозиторий `EXTRAS`.

Теперь отредактируем файл `/etc/drbd.conf` и создадим ресурс `rpgdata0`:

```
resource rpgdata0 {
  device /dev/drbd0;
  disk /dev/local/rpgdata0;
  meta-disk internal;
  on <host1> { address <адрес хоста 1>:<порт>; }
  on <host2> { address <адрес хоста 2>:<порт>; }
}
```

Замените имена `host1`, `host2`, `адрес хоста 1` и `адрес хоста 2` именами хостов и соответствующими сетевыми адресами.

Прежде чем переходить к следующему разделу, скопируйте файл `drbd.conf` на оба узла. Отключите автоматический запуск DRBD, потому что эту обязанность мы возложим на Pacemaker:

```
$ chkconfig drbd off
```

Включение ресурсов в DRBD

Перед инициализацией службы DRBD требуется также выполнить дополнительные настройки на стороне сервера. В данном случае массу проблем может вызвать SELinux, поэтому в дистрибутиве RedHat 6.X лучше отключить SELinux.

Чтобы отключить систему SELinux, точнее перевести ее в разрешающий режим, нужно в конфигурационном файле `/etc/sysconfig/selinux` изменить параметр `SELINUX`, как показано ниже:

```
SELINUX=permissive
```

Это необходимо проделать на обоих узлах. Затем следует перезагрузить систему и проверить, было ли соответствующее состояние установлено, выполнив следующую команду от имени root:

```
# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          permissive
Policy version:                 24
Policy from config file:        targeted
```

Как видите, в `Current mode` был установлен режим `permissive` (разрешающий).

Теперь пришло время добавить правило в `iptables`, разрешающее подключение к порту `7788`. Для этого можно добавить следующую строку непосредственно в файл `/etc/sysconfig/iptables`:

```
-A INPUT -m stat -state NEW -m tcp -p tcp --dport 7788 -j ACCEPT
```

После этого нужно перезапустить `iptables`:

```
# service iptables restart
iptables: Setting chains to policy ACCEPT: nat mangle filte[ OK ]
iptables: Flushing firewall rules:                         [ OK ]
iptables: Unloading modules:                               [ OK ]
iptables: Applying firewall rules:                         [ OK ]
```

Далее скопируйте конфигурационный файл на все ваши узлы и на этом настройку SELinux и iptables можно считать завершенной. Теперь инициализируем устройство и создадим необходимые метаданные.

Процедура инициализации должна выполняться на обоих узлах и от имени root:

```
$ drbdadm create-md rpgdata0
v08 Magic number not found
Writing meta data...
initialising activity log
NOT initializing bitmap
New drbd meta data block successfully created.
```

Это – процедура инициализации и должна выполняться только для новых устройств.

Далее включим устройство `rpgdata0`:

```
$ drbdadm up rpgdata0
```

Процесс включения можно проконтролировать наблюдением за виртуальной файловой системой `/proc`:

```
$ tail /proc/drbd
version: 8.4.1 (api:1/proto:86-100)
GIT-hash: 91b4c048c1a0e06837625f65d312b38d47abara80 build by buildsystem@
```

```
linbit, 2013-02-20 12:58:48
  0: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r-----
  ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:524236
```

Состояние `Inconsistent/Inconsistent` в данном случае – это нормально. Вы должны определить, какой узел будет ведущим и какой – источником синхронизации.

В настоящий момент у нас создан диск DRBD, настроена сеть и все готово к синхронизации.

Определение первичного устройства DRDB

Назначение первичного устройства выполняется просто; нужно перейти на первичный узел и выполнить следующую команду:

```
$ drbdadm primary rpgdata0
```

В результате сервер, на котором выполнена эта команда, станет мастером сервера репликации, и вы сможете создать физический том на этом новом устройстве. Для этого на ведущем узле (мастере) выполните следующую команду:

```
$ pvcreate /dev/drbd0
Physical volume "/dev/drbd0" successfully created
```

Создайте группу томов, в данном примере с именем `secured_vg_pg`:

```
$ vgcreate secured_vg_pg /dev/drbd0
Volume group "secured_vg_pg" successfully creatced
```

Наконец, на физическом томе можно создать логический том:

```
$ lvcreate -L 6G -n secured_lv_pg secured_vg_pg
```

В этом примере мы зарезервировали пространство для мгновенных снимков; поэтому, если они вам понадобятся, у вас будет достаточно пространства. В заключение можно настроить файловую систему.

Создание файловой системы на устройстве DRBD

Теперь важно проверить, исключена ли служба DRBD из списков запуска и остановки, потому что управление ею будет осуществляться с помощью Pacemaker. После отключения автоматического запуска и остановки службы можно приступать к созданию файловой системы на новом устройстве, но перед этим необходимо:

- создать точку монтирования;
- создать файловую систему;
- смонтировать файловую систему и открыть доступ к ней.

Вы можете создать свою точку монтирования, но в данном примере для этой цели будет использоваться каталог `/db/pgdata`:

```
$ mkdir -p -m 0700 /db/pgdata
```

Большинство дистрибутивов Linux поддерживает множество самых разных файловых систем; RedHat 6.0 имеет полноценную поддержку XFS. Файловая

система XFS обладает важным для нас свойством – поддержкой параллельного доступа. Она допускает параллельное выполнение операций чтения/записи. XFS позволяет нескольким потокам одновременно осуществлять запись в одни и те же файлы. Совершенно понятно, что это большое преимущество для больших таблиц в базе данных, помогающее избежать конкурентной борьбы между процессами.

Установить XFS и соответствующие утилиты можно командой:

```
$ yum install xfsprogs
```

XFS позволяет нескольким потокам одновременно осуществлять запись в одни и те же файлы; это особенно ценное свойство при использовании устройств DRBD, для которых конкурентная борьба за доступ к файловой системе является важным фактором.

После установки поддержки XFS можно отформатировать логический том командой:

```
$ mkfs.xfs /dev/secured_vg_pg/secured_lv_pg
```

После создания файловой системы ее размер нельзя уменьшить, зато можно увеличить, для чего существует утилита `xfs_growfs`.

Теперь можно смонтировать файловую систему:

```
$ mount -t xfs -o noatime,nodiratime,attr2 /dev/secured_vg_pg/secured_lv_pg /db/pgdata
```

Не забудьте настроить автоматическое монтирование этого раздела (в файле `fstab`); в противном случае он потеряется после перезагрузки.

и определить права доступа к каталогу, назначив владельцем пользователя PostgreSQL, обычно `postgres`:

```
$ chown postgres:postgres /db/pgdata
$ chmod 0700 /db/pgdata
```

Создание файловой системы должно выполняться только на первичном узле.

Теперь файловая система отформатирована, смонтирована и готова к использованию СУБД PostgreSQL.

Кластеры Pacemaker – интеграция с DRBD

Pacemaker превращает DRBD в чрезвычайно мощный инструмент в самых разных сценариях. Однако следует особое внимание обратить на некоторые моменты, которые уже рассматривались при обсуждении Pacemaker/Corosync:

- отключить механизм STONITH;
- отключить кворум;
- включить привязку (stickiness).

Как уже обсуждалось выше, очень важно избежать неправильного распределения полномочий между узлами и вероятных клинчей STONITH. Отключить STONITH можно командой:

```
$ crm configure property stonith-enabled="false"
```

Так как в данном случае мы снова имеем дело с кластером, состоящим из двух узлов, необходимо отключить кворум:

```
$ crm configure property no-quorum-policy=ignore
```

Теперь желательно определить степень связи ресурса с узлом. Необходимость этого действия уже обсуждалась выше. Тем не менее напомню, что, устанавливая степень связи, мы определяем предпочтительный узел для размещения ресурса. Это поможет избежать ненужных накладных расходов и простоев. Делается это с помощью команды:

```
$ crm configure property default-resource-stickiness="100"
```

Настройка включения DRBD

В этом разделе рассказывается, как запустить службу DRBD в кластере, работающем под управлением Pacemaker. Для этого нужно выполнить следующие действия:

- добавить DRDB в Pacemaker;
- добавить и определить ведущий/ведомый ресурс.

Нам необходимо определить ведущий/ведомый ресурс, определяющий, какой узел будет ведущим, а какой – ведомым. Сделать это можно следующей командой:

```
$ crm configure primitive drbd_pg ocf:linbit:drbd \
params drbd_resource="rpgdata0" \
op monitor interval="15" \
op start interval="0" timeout="240" \
op stop interval="0" timeout="120"
```

Затем следует настроить ресурс, который сможет повышать (promote) или понижать (demote) полномочия службы DRBD на каждом узле. Имейте в виду, что служба должна выполняться на обоих узлах одновременно, но действовать в разных состояниях, для этого определим ресурс ведущий/ведомый, как показано ниже:

```
$ crm configure ms ms_drbd_pg drbd_pg \
meta master-max="1" master-node-max="1" clone-max="2" \
clone-node-max="1" notify="true"
```

Pacemaker – настройка LVM

Теперь необходимо настроить в Pacemaker:

- управление LVM;
- управление файловой системой.

Вследствие особенностей работы DRBD фактический активный том будет невидим на вторичном (ведомом) узле. На вторичном узле нельзя смонтировать этот том. Поэтому требуется помочь DRBD найти устройства:

```
$ crm configure primitive pg_lvm ocf:heartbeat:LVM \
params volgrpname="secured_vg_pg" \
op start interval="0" timeout="30" \
op stop interval="0" timeout="30"
```

С учетом предыдущих настроек Pacemaker найдет пригодный для использования том на устройствах DRBD и сделает его доступным путем повышения полномочий ресурса DRBD. Учитывая, что в данном примере используется файловая система XFS, определим порядок монтирования и обслуживания данного устройства:

```
$ crm configure primitive pg_fs ocf:heartbeat:Filesystem \
params device="/dev/secured_vg_pg/secured_lv_pg" \
directory="/db/pgdata" \
options="noatime,nodiratime" fstype="xfs" \
op start interval="0" timeout="60" \
op stop interval="0" timeout="120"
```

Так как LVM является последним уровнем в этой конфигурации, можно воспользоваться возможностью создания мгновенных снимков и получить отличный уровень изоляции.

Pacemaker – настройка PostgreSQL

Теперь можно настроить PostgreSQL для работы в кластере.

Установка PostgreSQL не рассматривается здесь, потому что она уже обсуждалась в главе 1 «Развертывание Zabbix».

Следующая команда добавит в Pacemaker примитив, который будет проверять состояние PostgreSQL каждые 30 секунд, и определит предельное время ожидания ответа, равное 60 секундам:

```
$ crm configure primitive pg_lsb lsb:postgresql \
op monitor interval="30" timeout="60" \
op start interval="0" timeout="60" \
op stop interval="0" timeout="60"
```

Эта команда определяет увеличенный тайм-аут `start` и `stop`, потому что предполагается использование большой базы данных. Это необходимо, чтобы дать диспетчеру Pacemaker время завершить проверки при остановке и запуске.

Pacemaker использует эти параметры для проверки доступности PostgreSQL.

Pacemaker – настройка сети

До настоящего момента мы еще не определили предопределенный IP-адрес для PostgreSQL. Так как нежелательно, чтобы IP-адрес изменялся в случае переклю-

чения узла или отказа, нужно настроить виртуальный IP-адрес, который будет следовать за службой. Это предотвратит необходимость изменения конфигурации для всех ваших клиентов. Сделать это можно с помощью следующей команды:

```
$ crm configure primitive pg_vip ocf:heartbeat:IPaddr2 \
params ip="192.168.124.3" iflabel="pgvip" \
op monitor interval="5"
```

Обратите внимание: замените адрес `192.168.124.3` своим собственным.

Здесь указано, что ARP-адрес `IPaddr2` автоматически пошлет пять ARP-пакетов. Это значение можно увеличить при необходимости.

Pacemaker – заключительные настройки

Теперь у нас есть все необходимые компоненты, и их можно объединить в группу, содержащую все ресурсы. Назовем группу `PGServer`:

```
$ crm configure group PGServer pg_lvm pg_fs pg_lsb pg_vip
$ crm configure colocation col_pg_drbd inf: PGServer ms_drbd_pg:Master
```

Сервер `Master` указывает, что группа `PGServer` зависит от установки статуса ведущего, которое присваивается исключительно активному узлу. Группа `PGServer` также зависит от установки статуса ведущего DRBD.

Теперь важно определить правильный порядок запуска и остановки служб:

```
$ crm configure order ord_pg inf: ms_drbd_pg:promote PGServer:start
```

Параметры `:promote` и `:start` являются основными; они определяют, что одновременно с повышением полномочий `ms_drdb_pg` должна запускаться группа `PGServer`. Если опустить параметр `:start`, Pacemaker будет сам определять порядок запуска/остановки, что может привести к появлению неисправностей.

Настройка кластера – заключительная проверка

Наконец-то кластер готов! Что дальше? Все просто! Вы можете попробовать изменять настройки своего кластера, поиграть с ним и убедиться, что все прекрасно, прежде чем запустить в работу эту новую инфраструктуру.

Прежде всего нужно проверить реакцию кластера на следующие ситуации:

- отключение от сети одного узла;
- переключение ресурсов вручную;
- крах ведущего узла;
- крах ведомого узла;
- принудительная синхронизация всех данных.

Выполните следующую команду:

```
$ crm node standby HA-node2
```

Если все хорошо, `crm_mon` сообщит следующее:

```
Node HA-node2: standby
Online: [ HA-node1 ]
```

Эту ситуацию легко исправить командой:

```
$ crm node online HA-node2
```

Пока нет никаких сложностей. Теперь попробуйте вручную переключить ресурсы всего кластера:

```
$ crm resource migrate PGServer HA-node2
```

Вы можете переместить `PGServer` на второй узел. Если он окажется недоступным, Pacemaker переместится на первичный узел, пока вторичный не станет доступным вновь. Это объясняется тем, что команда `migrate` присваивает более высокий рейтинг указанному узлу, который имеет более высокий приоритет перед степенью связи (stickness), установленной вами.

Вернуть сервер обратно на первичный узел можно командой:

```
$ crm resource unmigrate PGServer
```

Теперь можно попробовать выключить вторичный узел, и Pacemaker сообщит:

```
Online: [ HA-node1 ]
OFFLINE: [ HA-node2 ]
Master/Slave Set: ms_drbd_pg [drbd_pg]
Masters: [ HA-node1 ]
Stopped: [ drbd_pg:1 ]
```

После этого запустите вторичный узел, и Pacemaker сообщит:

```
Online: [ HA-node1 HA-node2 ]
Master/Slave Set: ms_drbd_pg [drbd_pg]
Masters: [ HA-node1 ]
Slaves: [ HA-node2 ]
```

Теперь последний тест: следующей командой можно объявить недействительными все данные на вторичном узле:

```
$ drbdadm invalidate-remote all
```

Тот же эффект можно получить, выполнив следующую команду на вторичном узле:

```
$ drbdadm invalidate all
```

В результате механизм DRBD решит, что все данные на вторичном узле оказались рассинхронизированными, и выполнит их синхронизацию с первичным узлом.

Производительность и оптимизация DRBD

Кластер на основе DRBD обладает определенными аспектами, которые можно улучшить, и это обязательно следует учитывать при реализации. В данном случае имеется несколько оптимизаций, которые можно применить. Представьте, что база данных, точнее вторичный узел в кластере DRBD, географически удален от

вашего вычислительного центра и пропускная способность сети, необходимая для эффективной синхронизации, играет важную роль. В этом случае важно рассчитать, какой объем данных потребуется переместить и какую скорость должна обеспечить сеть в аварийной ситуации.

Эффективная синхронизация DRBD

Синхронизация – это особый случай, и она не может рассматриваться в одном ряду с репликацией устройства. Репликация выполняется только в момент первого запуска устройства, а синхронизация производится всякий раз, когда осуществляется запись. В предлагаемой архитектуре синхронизация необходима, когда:

- была нарушена связь;
- произошел отказ первичного узла;
- произошел отказ вторичного узла.

DRBD синхронизирует блоки не последовательно и не в том порядке, в каком они первоначально были записаны.

В процессе синхронизации на диске будут иметься частично устаревшие и частично обновленные данные.

Служба продолжает выполняться на первичном узле, пока на вторичном производится фоновая синхронизация. Поскольку архитектура включает уровень LVM, лежащий поверх DRBD, в ходе синхронизации можно использовать мгновенные снимки; это мощная особенность данной архитектуры. Пока продолжается процесс синхронизации, надежность системы ухудшается; достоверную информацию содержит только первичный узел, и если с ним что-то произойдет, можно потерять все данные, и вторичный узел останется с недостоверными данными. Эту критическую ситуацию можно смягчить за счет использования мгновенных снимков LVM.

Создание мгновенного снимка перед началом синхронизации может дать вам практический опыт преодоления подобных ситуаций, потому что данные на вторичном узле останутся непротиворечивыми и достоверными, хотя и не самыми свежими. Создание снимков перед синхронизацией уменьшит расчетное время восстановления (Estimated Time to Repair, ETR), известное также как директивное время восстановления (Recovery Time Objective, RTO).

Чтобы автоматизировать создание мгновенных снимков, добавьте следующие строки в конфигурацию DRBD:

```
resource RESOURCE_NAME {
  handlers {
    before-resync-target "/usr/lib/drbd/snapshot-resync-target-lvm.sh";
    after-resync-target "/usr/lib/drbd/unsnapshot-resync-target-lvm.sh";
  }
}
```

Эти настройки предполагают запуск сценария `snapshot-resync-target-lvm.sh` для создания мгновенного снимка перед началом синхронизации и сценария `unsnapshot-resync-target-lvm.sh` для удаления снимка по ее завершении.

Если сценарий завершится с ошибкой, синхронизация не начнется.

Наиболее оптимальной для DRBD считается синхронизация по контрольным суммам. Такой вид синхронизации эффективнее, так как запись выполняется целыми блоками, что запрещено по умолчанию. В этом режиме DRBD читает одни и те же блоки на ведущем и ведомом узлах перед синхронизацией и вычисляет контрольные суммы по их содержимому. Если контрольные суммы совпадают, такие блоки пропускаются как не требующие синхронизации.

Чтобы установить этот режим, добавьте следующие строки в конфигурационный файл DRBD:

```
resource <resource>
  net {
    csums-alg <algorithm>;
  }
  ...
}
```

Тег `<algorithm>` – это название любого алгоритма хэширования, поддерживаемого криптографическим API ядра, обычно один из: `sha1`, `md5` и `crc32c`.

Если это изменение производится для существующего ресурса, вам нужно скопировать файл `drbd.conf` на второй узел и затем выполнить команду:

```
drbdadm adjust <resource>
```

Включение онлайн-верификации для DRBD

Онлайн-верификация – очень эффективный способ поблочной проверки целостности данных. Он особенно интересен с точки зрения эффективного использования полосы пропускания; кроме того, он не нарушает избыточности.

Онлайн-верификация оказывает значительную нагрузку на центральный процессор.

В этом режиме DRDB вычисляет криптографический дайджест всех блоков на первом узле и посылает их второму узлу, который выполняет ту же проверку. Если дайджесты отличаются, блок маркируется как рассинхронизированный, и DRBD пересылает только их. Этот режим выключен по умолчанию. Чтобы включить его, добавьте следующие строки в `drbd.conf`:

```
resource <resource>
  net {
    verify-alg <algorithm>;
  }
  ...
}
```

Здесь вместо `<algotithm>` также следует указать название любого алгоритма хэширования, поддерживаемого криптографическим API ядра, обычно один из: `sha1`, `md5` и `crc32c`. После настройки онлайн-верификацию можно выполнить командой

```
$ drbdadm verify <resource>
```

Поскольку проверка гарантирует полную синхронизацию узлов, рекомендуется выполнять ее в специально запланированные дни, раз в неделю или в месяц, с помощью планировщика `crontab`.

Если появились рассинхронизированные блоки, их можно синхронизировать простой командой:

```
$ drbdadm disconnect <resource>
$ drbdadm connect <resource>
```

DRBD – некоторые аспекты настройки сети

При использовании блочной файловой системы на DRBD можно ускорить скорость передачи, увеличив **максимальный размер блока передачи (Maximum Transmission Unit, MTU)**.

Блочные файловые системы, такие как EXT3, ReiserFS и GFS, имеют большое преимущество. Файловая система XFS, предложенная в этом примере, основана на использовании экстентов (непрерывных областей), поэтому для нее увеличение размера кадра не даст большого увеличения производительности.

DRBD позволяет настраивать норму синхронизации. Обычно синхронизация вторичного узла выполняется как можно скорее, чтобы уменьшить отрезок времени, в течение которого данные находятся в непоследовательном состоянии. Однако часто бывает желательно предотвратить снижение производительности из-за расходования значительной доли пропускной способности на синхронизацию.

Обязательно проверьте соответствие этого параметра имеющейся пропускной способности. Бессмысленно задавать высокую норму, если для ее поддержания недостаточно пропускной способности.

Максимальная пропускная способность, используемая фоновым процессом повторной синхронизации, ограничивается параметром со значением, выраженным в байтах; то есть 8192 означает 8 МиБ. Чтобы изменить норму, добавьте следующие строки в конфигурационный файл DRBD:

```
resource <resource>
  disk {
    resync-rate 50M;
    ...
  }
  ...
}
```

Обычно норма вычисляется по формуле: `MAX_ALLOWED_BANDWIDTH * 0.3`. То есть под нужды синхронизации отводится до 30% максимально доступной пропускной способности.

То же правило применяется для ограничения нормы синхронизации в файле `drbd.conf`:

```
resource <resource>
  syncer {
    rate 50M;
    ...
  }
  ...
}
```

Значение параметра `syncer rate` можно временно изменить командой

```
drbdsetup /dev/drbdnum syncer -r 120M
```

Значение параметра `resync-rate` можно временно изменить командой

```
drbdadm disk-options --resync-rate=110M <resource>
```

Вернуть значения из конфигурационного файла для обоих параметров можно командой

```
drbdadm adjust resource
```

DRBD имеет ряд интересных параметров для тонкой настройки системы и оптимизации производительности, но с их помощью невозможно решить все проблемы. В разных версиях они могут отличаться, но знать об их существовании полезно, а их применение может принести некоторые выгоды.

В частности, есть два параметра:

- `max-buffers`;
- `max-epoch-time`.

Первый (`max-buffers`) определяет максимальное количество буферов DRBD. Второй (`max-epoch-time`) определяет максимальное количество запросов на запись между двумя барьерами записи. Оба можно изменить в файле `drbd.conf`:

```
resource <resource> {
  net {
    max-buffers 8000;
    max-epoch-size 8000;
    ...
  }
  ...
}
```

Оба параметра имеют значение по умолчанию `2048`, и обоим можно увеличить его до `8000`. Это вполне допустимое значение для современных контроллеров RAID-SCSI.

Ниже описывается еще одна оптимизация, имеющая отношение к сети, – размер буфера передачи для протокола TCP/IP. По умолчанию этот параметр имеет значение 128 КБ, но если в вашем распоряжении имеется сеть с высокой пропускной способностью, имеет смысл увеличить это значение до 512 КБ.

```
resource <resource> {
  net {
```

```
    sndbuf-size 512K;
    ...
  }
  ...
}
```

Если присвоить этому параметру значение `0`, DRBD определит оптимальное значение автоматически, адаптируя буфер TCP под сеть.

В завершение темы оптимизации хочется также сказать, что DRBD имеет еще следующие параметры настройки:

- `no-disk-barrier`;
- `no-disk-flushes`;
- `no-disk-drain`.

Но лично я советую настраивать их только тем, кто имеет полное представление об имеющемся оборудовании. Эти параметры отключают барьеры записи (write barriers), сброс на диск (disk flush) и очистку (drain). Обычно всеми этими функциями управляет непосредственно контроллер, поэтому нет большого смысла возлагать эту обязанность на DRBD.

В заключение

В этой главе вы познакомились с некоторыми фундаментальными идеями высокой доступности и установки служб на кластерах. Вы также узнали, как применить эти идеи к архитектуре Zabbix с помощью открытого диспетчера ресурсов Pacemaker и механизма DRBD репликации файловой системы. На протяжении всей главы разъяснялось, почему следует сохранить архитектуру максимально простой и легковесной и использовать минимально возможное число узлов, необходимое для обеспечения надежности.

Эта глава завершает первую часть книги, основной целью которой было помочь выбрать оптимальное решение Zabbix для окружений любого размера. После выбора нужного оборудования, поддерживающего программного обеспечения (см. *главу 2 «Распределенный мониторинг»*) и настройки высокой доступности для наиболее важных компонентов у вас теперь должна иметься подготовленная система Zabbix, отлично приспособленная для вашего окружения и отвечающая вашим потребностям.

В оставшейся части книги мы сосредоточимся на использовании этой конфигурации для организации мониторинга вашей сети и серверов и применении собранных данных для чего-нибудь посложнее, чем простой вывод предупреждений. В следующей главе описываются большинство встроенных типов элементов Zabbix и приемы сбора данных мониторинга из множества простых, сложных и смешанных источников.

Глава 4

Сбор данных

Теперь, когда у вас имеется инфраструктура Zabbix, подготовленная для вашего окружения, можно попробовать использовать ее для мониторинга. Несмотря на простоту идентификации хостов и устройств, физических или логических, подлежащих мониторингу, в настоящий момент может быть не совсем очевидно, какие фактические параметры можно измерять. Измеряемые параметры называются элементами (items), и эта глава обсуждает их основные особенности и характеристики. Первая ее часть будет посвящена в основном теоретическим положениям:

- элементы как измеряемые параметры;
- движение данных и направленность элементов;
- элементы-ловушки (трапперы) как средство управления движением данных.

Затем мы перейдем к практической части и конкретным подходам и обсудим настройку элементов для сбора данных из следующих источников:

- базы данных и источники ODBC;
- Java-приложения, консоль JMX и агенты SNMP;
- SSH-мониторинг;
- элементы IMPI;
- мониторинг веб-страниц;
- смешанные и вычисляемые элементы.

Сбор простых данных

Одной из важнейших особенностей, отличающих Zabbix от большинства других решений мониторинга, является режим взаимодействий с подконтрольными объектами, суть которого заключается в сборе простых данных, а не в обновлении состояния или рассылке оповещений. Иными словами, многие приложения мониторинга действуют по схеме, представленной на рис. 4.1.

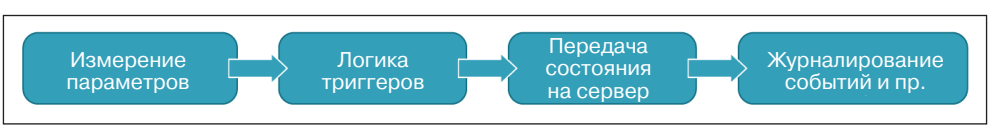


Рис. 4.1 ❖ Порядок работы большинства приложений мониторинга

То есть агент или другое средство мониторинга не только производит измерения, но также принимает решение о состоянии контролируемого объекта по результатам измерений и посылает выводы главному серверному компоненту для дальнейшей обработки.

Порядок работы Zabbix кардинально отличается, как показано на рис. 4.2.

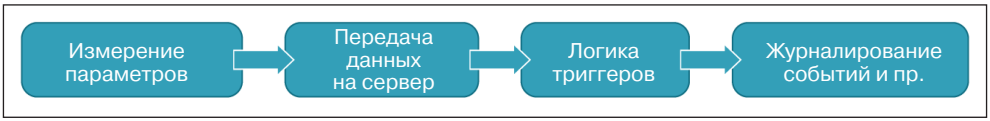


Рис. 4.2 ❖ Порядок работы системы мониторинга Zabbix

Здесь агент выполняет только простые измерения и посылает результаты серверному компоненту для сохранения и дальнейшей обработки.

Возвращаемые агентом данные не связаны с решением, принимаемым триггером (норма/отказ или какое-то другое), а просто сохраняются на сервере в виде простых результатов измерений. Данные, к которым это применимо, то есть числовые данные, дополнительно агрегируются и сохраняются в виде трендов (минимальное, максимальное, среднее) за разные периоды времени. Отделение данных от логики принятия решений и сохранение их в центральном хранилище дают системе Zabbix два важных преимущества.

Первое: возможность использовать Zabbix для сбора данных о чем-то, не связанном непосредственно с оповещением и действиями, которые могут предприниматься в ответ, но имеющем отношение к общей производительности и работоспособности системы. Классическим примером является сетевой коммутатор со множеством портов. Возможно, вам неинтересно получать оповещения об аномальном трафике, протекающем через каждый отдельный порт (так как довольно сложно определить аномальность трафика, не имея контекстной информации), но интересно иметь информацию об объеме трафика, протекающего через каждый отдельный порт и весь коммутатор в целом, чтобы получить общее представление о ситуации, определить узкие места или спланировать дальнейшее развитие сетевой инфраструктуры. То же относится к расходованию процессорного времени и каждого из его ядер в отдельности, емкости хранилища, количеству пользователей, одновременно работающих с заданным приложением, и многому другому. В простейшем случае систему Zabbix можно использовать лишь для сбора данных и их визуализации в виде комплекса графиков и диаграмм, не применяя триггеров и других инструментов, и тем не менее оправдать затраты времени и ресурсов на ее создание.

Второе важное преимущество наличия полноценной централизованной базы данных с исходной информацией, а не только единичного замера (или, точнее, нескольких измерений одного и того же элемента), состоит в том, что триггеры и логика принятия решений имеют все, что им необходимо. Имея целую базу данных с замерами, можно точно определять виды событий и отправлять оповещения. Нет необходимости полагаться на единственное измерение, как нет необходимости по-

лагаться на последнее и несколько предыдущих измерений или ограничиваться элементами с одного хоста. Фактически перед вами открывается возможность находить корреляции с любыми другими элементами в истории. Это настолько мощная возможность, что ей будет посвящена отдельная глава (*глава 5* «*Визуализация данных*»), и вы можете прямо сейчас перейти к ней, если эта тема вам более интересна в данный момент. Вся эта мощь обусловлена отделением функций сбора данных от логики триггеров и принятия решений. Измерения в Zabbix – это только измерения, и ничего больше.

Итак, элемент данных в Zabbix представляет один измеряемый параметр – единственный источник данных. В Zabbix поддерживается много встроенных типов элементов, помимо возможности определять пользовательские типы. В этой главе вы познакомитесь также с некоторыми неочевидными, но очень интересными типами. Вы увидите, как организовать работу с базами данных, как интегрировать с Zabbix сторонние ловушки SNMP, как объединять имеющиеся элементы для представления и мониторинга кластера и многое другое. Научившись определять элементы и собирать данные, вы заложите прочный фундамент и на его основе сможете построить свои механизмы управления событиями и визуализации данных, о которых рассказывается в последующих главах.

Потоки данных и элементы

Элемент Zabbix определяется такими характеристиками, как идентификатор, тип данных и хост. Все эти характеристики обычно более полезны для других компонентов Zabbix. Идентификатор (обычно это имя и ключ элемента) и хост используются для различения среди тысяч других элементов, определенных в системе мониторинга. Тип данных необходим системе Zabbix, чтобы знать, как хранить элемент, как его визуализировать (текстовые данные, например, не могут использоваться для построения графиков) и, самое важное, какие функции можно применять для обработки элемента.

Имя элемента – это описательная метка, легко читаемая человеком, тогда как ключ соответствует специальному синтаксису и точно определяет измеряемый параметр.

Две другие очень важные характеристики, присущие всем элементам данных, – это продолжительность хранения в истории (и трендах) и тип элемента. В *главе 1* «*Развертывание Zabbix*» вы уже видели, как длительность хранения истории влияет на размер базы данных мониторинга, как оценить ее возможный размер и как найти баланс между производительностью и доступностью данных. Тип элемента, в свою очередь, определяет, как Zabbix будет хранить данные и как собирать их: с помощью агента, посредством запросов SNMP, с применением внешних сценариев или как-то иначе.

Как вы, возможно, уже знаете, существует большое разнообразие типов элементов. Вы без труда разберетесь в различиях между элементами SSH и ODBC, но для вас важно понять, как данные передаются между сервером и средствами измерения,

такими как агенты Zabbix или внешние сценарии. С этой целью мы сначала сосредоточимся на агентах Zabbix и различиях между пассивными и активными элементами.

Понятия «пассивный» и «активный» в первую очередь относятся к агенту, а не к серверу. Кроме того, под активным подразумевается тот компонент, который инициализирует соединение для передачи или получения конфигурационной информации и мониторинга данных, как показано на рис. 4.3.

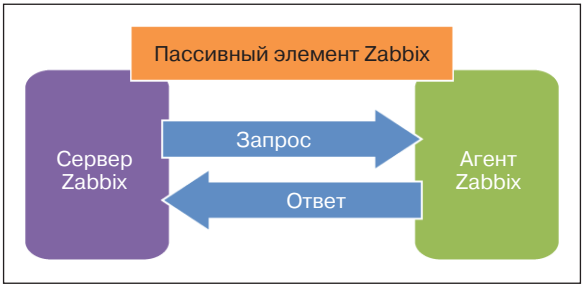


Рис. 4.3 ❖ Пассивный элемент Zabbix

С точки зрения агента стандартный элемент Zabbix является пассивным. Это означает, что сервер сам определяет интервал и посылает запросы агенту, чтобы получить желаемый элемент. Для этого сервер инициирует сетевое соединение и разрывает его, а агент только принимает входящие запросы.

В случае с активным элементом, напротив, агент запрашивает у сервера интервал и какие данные тот желает получить. Затем он планирует порядок выполнения измерений и инициирует соединение с сервером, чтобы послать результаты для дальнейшей обработки. С точки зрения сетевых операций, в этом случае устанавливаются два сеанса (как показано на рис. 4.4):

- агент запрашивает у сервера информацию об элементах и интервалах измерений;
- агент посылает результаты измерений серверу.

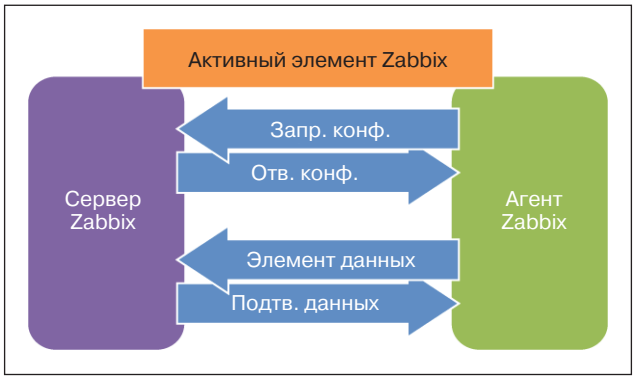


Рис. 4.4 ❖ Активный элемент Zabbix

Для обслуживания активных элементов агент должен настраиваться так, чтобы он знал, к какому серверу обращаться для получения конфигурации и передачи данных. Все это определяется в конфигурационном файле `zabbix_agentd.conf` для каждого агента; в параметрах `ServerActive` (определяющем имя хоста или IP-адрес сервера Zabbix) и `RefreshActiveChecks` (определяющем количество секунд, через которое агент должен вновь запросить информацию об измеряемых параметрах).

Кроме инициатора сетевых соединений, главное отличие между пассивным и активным элементами состоит в том, что в последнем случае отсутствует возможность гибко управлять интервалами между измерениями. Для пассивного элемента можно определять разные интервалы в зависимости от времени суток и дня недели. Например, доступность сервера идентификации пользователей в рабочие часы можно проверять раз в минуту, а в нерабочие – раз в десять минут. При использовании активных элементов, напротив, измерения выполняются с четко установленным интервалами.

Обратите также внимание на разительное сходство различий между активными и пассивными элементами и между активными и пассивными прокси-серверами.

Фактически, выбирая между активными и пассивными элементами, можно руководствоваться теми же причинами и соображениями, что и при выборе между активными и пассивными прокси-серверами, как описывалось в *главе 2 «Распределенный мониторинг»*, чтобы снять с сервера часть нагрузки, связанной с планированием заданий, и обойти некоторые ограничения, действующие в сети.

Впрочем, между прокси-серверами и агентами есть одно важное отличие. Прокси-сервер может собирать данные мониторинга с нескольких хостов, тогда как агент теоретически (но не фактически, учитывая возможность использовать нестандартные элементы, опирающиеся на внешние сценарии или приложения) ограничивается мониторингом только того хоста, на котором он установлен.

Еще одно существенное отличие обнаруживается, когда речь заходит о движении данных: режим работы прокси применяется ко всем хостам и элементам, которые он обслуживает. Фактически прокси-сервер не учитывает природу элементов. Однако когда данные собираются активным прокси-сервером (с помощью активных или пассивных агентов, внешних сценариев, SNMP, SSH и т. д.), соединение с основным сервером всегда инициирует прокси. Соответственно, действует пассивный прокси-сервер; его не интересует активная/пассивная природа агентов или элементов. Он всегда ждет изменений в конфигурации от основного сервера и запросов на передачу измеренных параметров.

Элемент, активный или пассивный, – лишь один из множества. Для любого хоста может быть определена смесь активных и пассивных элементов; поэтому не следует надеяться, что агент всегда будет инициировать соединения с сервером для передачи любых элементов. Чтобы обеспечить передачу всех данных активным агентом, все элементы данных должны определяться как активные, в том числе и будущие.

Ловушки элементов Zabbix

Ловушки (трапперы) Zabbix – особая разновидность активного элемента, использующего протокол агентов Zabbix. В отличие от всех остальных типов элементов, для элемента-ловушки, или элемента-траппера, не определяется интервал мониторинга на уровне сервера. Другими словами, сервер знает о существовании элемента-траппера Zabbix, его тип данных, хост, с которым он связан, и длительность хранения в истории и трендах. Но он никогда не планирует проверку элемента и не передает информацию с интервалом мониторинга прокси-серверам или агентам. То есть логика проверки сама должна определять интервал и посылать информацию о собранных данных на сервер.

В некотором смысле элементы-трапперы являются противоположностью внешних проверок. Как вы, возможно, уже знаете, элемент внешней проверки создается, когда для получения измерений требуется, чтобы сервер выполнил внешний сценарий. Это может стать причиной падения производительности сервера из-за необходимости запускать новый процесс для выполнения каждого внешнего сценария и последующего ожидания ответа. Увеличение количества внешних сценариев может существенно снижать быстродействие и приводить к накоплению просроченных проверок. Чрезвычайно простое и эффективное решение этой проблемы (кроме уменьшения количества внешних сценариев до необходимого минимума) заключается в преобразовании всех элементов внешних проверок в трапперы, планировании выполнения тех же сценариев с использованием `crontab` или любого другого планировщика и изменении самих сценариев так, чтобы для передачи данных они использовали команду `zabbix_sender`. Когда мы будем обсуждать протокол Zabbix в *главе 8* «*Внешние сценарии*», вы увидите довольно много примеров применения данного решения.

Потоки данных

В этом разделе приводится краткое описание типов элементов, сгруппированных по типу соединения, с возможными альтернативами, если они вдруг понадобятся по каким-то причинам. Как нетрудно догадаться, трапперы Zabbix часто оказываются единственной, хотя и громоздкой, альтернативой, когда необходимо изменить тип соединения на обратный. Обратите внимание, что в табл. 4.1 термин «пассивный» означает инициализацию соединения сервером, а «активный» – выполняемой проверкой. На первый взгляд, это противоречит здравому смыслу, но на самом деле все зависит от точки зрения, как в случае с прокси и агентами.

Таблица 4.1 ❖ Типы элементов

Тип элемента	Направление	Альтернатива
Агент Zabbix	Пассивный	Агент Zabbix (активный)
Агент Zabbix (активный)	Активный	Агент Zabbix
Простая проверка	Пассивный	Траппер Zabbix
Агент SNMP	Активный	Не применимо к данному случаю

Таблица 4.1 *(окончание)*

Тип элемента	Направление	Альтернатива
Внутренний	Не применимо к данному случаю (данные мониторинга самого сервера)	Не применимо к данному случаю
Траппер Zabbix	Активный	Зависит от природы данных
Агрегат Zabbix	Не применимо к данному случаю (используются данные, уже хранящиеся в базе данных)	Не применимо к данному случаю
Внешняя проверка	Пассивный	Траппер Zabbix
Монитор базы данных	Пассивный	Траппер Zabbix
Агент IPMI	Пассивный	Траппер Zabbix
Агент SSH	Пассивный	Траппер Zabbix
Агент TELNET	Пассивный	Траппер Zabbix
Агент JMX	Пассивный	Траппер Zabbix
Dsxbckztvsq	Не применимо к данному случаю (используются данные, уже хранящиеся в базе данных)	Не применимо к данному случаю

В следующих разделах мы познакомимся с еще более сложными и интересными типами элементов.

Мониторинг базы данных с помощью Zabbix

Zabbix позволяет посылать SQL-запросы любой базе данных. Результат, возвращаемый базой данных, сохраняется как значение элемента, который, в свою очередь, может иметь триггеры. Эта возможность может пригодиться для самых разных целей, в частности с ее помощью можно следить за количеством пользователей, в настоящий момент подключенных к базе данных и к веб-порталу, или просто извлекать данные мониторинга.

ODBC

ODBC – это программный слой между **системой управления базами данных (СУБД)** и приложением. Приложение использует функции ODBC через специальный драйвер ODBC. Драйверы ODBC обычно реализуются и развиваются создателями СУБД, чтобы дать другим разработчикам возможность использовать их базы данных через этот слой. Конфигурационный файл определяет драйвер, который загружает все параметры соединения для каждого **имени источника данных (Data Source Name, DSN)**, и внутри этого файла перечисляются и определяются все DSN. Кроме того, DSN дает возможность представить всю базу данных в удобочитаемом формате. Файл DSN требует защиты. В предлагаемой конфигурации рекомендуется использовать отдельную учетную запись Unix для сервера Zabbix, что избавит вас от лишних сложностей. Так как имеется только один сервер Zabbix, достаточно открыть доступ к данному файлу только для его учетной записи. Владельцем файла должен быть этот пользователь, а для остальных данный файл

следует сделать недоступным для чтения. Имена источников данных (DSN) для всех используемых баз данных хранятся в файле /etc/odbc.ini. Позаботьтесь о его защите и закройте доступ к нему для всех других пользователей, потому что он содержит пароли.

В настоящее время доступны две открытые версии ODBC: **unixODBC** и **iODBC**. Zabbix может взаимодействовать с любой из них, но в предлагаемой конфигурации используется версия unixODBC. Установить ее можно двумя способами: с помощью диспетчера пакетов и старым добрым способом – из исходных текстов (в настоящее время текущей стабильной является версия 2.3.2):

```
$ wget ftp://ftp.unixodbc.org/pub/unixODBC/unixODBC-2.3.2.tar.gz
$ tar zxvf unixODBC-2.3.2.tar.gz
$ cd unixODBC-2.3.2
$ ./configure --prefix=/usr --sysconfdir=/etc
$ make
$ make install
```

Если вы используете 64-разрядную систему, укажите 64-разрядные версии библиотек в --libdir:

```
./configure --prefix=/usr --sysconfdir=/etc --libdir=/usr/lib64
```

Скомпилированные двоичные файлы по умолчанию устанавливаются в каталог /usr/bin, а библиотеки, в зависимости от версии, – в каталог /usr/lib или /usr/lib64.

Желающие установить unixODBC с помощью диспетчера пакетов могут воспользоваться командой

```
$ yum -y install unixODBC unixODBC-devel
```

Установка драйверов баз данных

unixODBC имеет обширный перечень поддерживаемых баз данных, в том числе:

- MySQL;
- PostgreSQL;
- Oracle;
- DB2;
- Sybase;
- Microsoft SQL Server (через FreeTDS).

Полный список баз данных, поддерживаемых unixODBC, можно найти по адресу: http://www.unixodbc.org/drivers.html.

Драйвер MySQL ODBC

Теперь, если вы установили unixODBC с помощью диспетчера пакетов, следуя той же процедуре, например в Red Hat, можно установить драйвер MySQL ODBC:

```
$ yum install mysql-connector-odbc
```

Чтобы установить драйвер из исходных текстов, загрузите пакет, например mysql-connector-odbc-5.1.13-linux-glibc2.5-x86-64bit.tar.gz.

Разархивируйте и скопируйте его содержимое в каталоги /usr/lib/odbc и /usr/lib64/odbc/:

```
$ tar xzf mysql-connector-odbc-5.1.13-linux-glibc2.5-x86-64bit.tar.gz
$ mkdir /usr/lib64/odbc/
$ cp /usr/src/ mysql-connector-odbc-5.1.13-linux-glibc2.5-x86-64bit/lib/* /usr/lib64/odbc/
```

После этого проверьте наличие всех необходимых библиотек в системе с помощью команды ldd. Для этого в 32-разрядной системе выполните команду

```
$ ldd /usr/lib /libmyodbc5.so
```

а 64-разрядной:

```
$ ldd /usr/lib64 /libmyodbc5.so
```

Если напротив элементов списка в выводе команды не появилось слов Not Found, это означает, что все необходимые библиотеки имеются и можно двигаться дальше; в противном случае установите отсутствующие библиотеки.

Все установленные драйверы ODBC перечислены в файле /etc/odbcinst.ini; для MySQL 5 в этом файле должны содержаться следующие строки:

```
[mysql]
Description = ODBC for MySQL
Driver      = /usr/lib/libmyodbc5.so
Setup       = /usr/lib/libodbcmyS.so
```

В 64-разрядной системе пути к библиотекам немного отличаются:

```
[mysql]
Description = ODBC for MySQL
Driver64    = /usr/lib64/libmyodbc5.so
Setup64     = /usr/lib64/libodbcmyS.so
```

По всем вопросам относительно драйвера MySQL ODBC обращайтесь к официальной документации: http://dev.mysql.com/doc/connector-odbc/en/.

Все источники данных определены в файле odbc.ini. Создайте этот файл и добавьте в него следующие строки:

```
[mysql-test]
# Используется имя драйвера, как указано в odbcinst.ini
Driver      = MySQL5
Description = Connector ODBC MySQL5
Database    = <имя базы данных>
USER        = <имя пользователя>
Password    = <пароль доступа к базе данных>
SERVER      = <IP-адрес>
PORT        = 3306
```

Есть возможность настроить в ODBC использование защищенного SSL-соединения, но для этого требуется сгенерировать сертификат и настроить обе стороны (ODBC и сервер). За более подробной информацией обращайтесь к официальной документации.

Драйвер PostgreSQL ODBC

Чтобы получить доступ к базе данных PostgreSQL через ODBC, нужно установить соответствующий драйвер. Он используется сервером Zabbix для отправки запросов в любые базы данных PostgreSQL по протоколу ODBC.

Официальный драйвер ODBC для PostgreSQL доступен по адресу: http://www.postgresql.org/ftp/odbc/versions/src/.

Для установки драйвера выполните следующие шаги:

1. Загрузите, скомпилируйте и установите драйвер psqlODBC:

```
$ tar -zxvf psqlodbc-xx.xx.xxxx.tar.gz
$ cd psqlodbc-xx.xx.xxxx
$ ./configure
$ make
$ make install
```

2. Сценарий `configure` имеет множество параметров; наиболее важные из них:

```
--with-libpq=<путь к каталогу установки>
--with-unixodbc=<путь к каталогу установки unixODBC или к файлу odbc_config>
--enable-pthreads #включает поддержку многопоточного выполнения, если доступна (не
для всех платформ)
```

3. При желании драйвер можно установить с помощью диспетчера пакетов:

```
$ yum install postgresql-odbc
```

4. После установки драйвера ODBC создайте файл `/etc/odbcinst.ini` со следующим содержимым (если установка выполнялась из пакета RPM, просто проверьте его):

```
[PostgreSQL]
Description = PostgreSQL driver for Linux
Driver      = /usr/local/lib/libodbcpsql.so
Setup       = /usr/local/lib/libodbcpsqlS.so
Driver64    = /usr/lib64/psqlodbc.so
Setup64     = /usr/lib64/libodbcpsqlS.so
```

5. Теперь вызовите утилиту `odbcinst` и передайте ей свой шаблон:

```
$ odbcinst -i -d -f template_filepsql
```

ODBC поддерживает авторизацию с шифрованием по алгоритму `md5`, с применением `crypt`. Имейте в виду, что шифрование выполняется только после авторизации. Все запросы ODBC посылает в виде открытого текста. Начиная с версии 08.01.002 `psqlODBC` поддерживает SSL-соединения, защищающие ваши данные.

6. Драйвер psqlODBC поддерживает многопоточный режим выполнения, поэтому для каждого драйвера можно определить уровень сериализации потоков. Например, в файл `odbcinst.ini` можно добавить следующую строку:

```
[PostgreSQL]
Description = PostgreSQL driver for Linux
Driver      = /usr/local/lib/libodbcpsql.so
Setup       = /usr/local/lib/libodbcpsqlS.so
Threading = 2
```

7. Теперь нужно настроить файл `odbc.ini`. Для этого можно использовать утилиту `odbcinst`, передав ей шаблон, или просто отредактировать его в текстовом редакторе:

```
$ odbcinst -i -s -f template_file
```

8. Файл `odbc.ini` должен содержать строки, напоминающие следующие:

```
[PostgreSQL]
Description      = Postgres to test
Driver           = /usr/local/lib/libodbcpsql.so
Trace            = Yes
TraceFile        = sql.log
Database         = <имя базы данных>
Servername       = <имя сервера или IP-адрес>
UserName         = <имя пользователя>
Password         = <пароль доступа к базе данных>
Port             = 5432
Protocol         = 6.4
ReadOnly         = No
RowVersioning    = No
ShowSystemTables = No
ShowOidColumn    = No
FakeOidIndex     = No
ConnSettings     =
```

Драйвер Oracle ODBC

Oracle – еще одна широко используемая СУБД, для которой также имеется драйвер ODBC. Ниже подробно описывается, как установить драйвер Oracle ODBC, потому что на сайте http://www.unixodbc.org приводится очень мало информации об этом.

1. Прежде всего необходимо на сайте Oracle получить клиента. Oracle распространяет несколько пакетов в виде rpm и tar.gz, как показывает следующая команда:

```
$ rpm -I oracle-instantclient11.2-basic-11.2.0.1.0-1.i386.rpm oracle-
instantclient11.2-odbc-11.2.0.1.0-1.i386.rpm oracle-instantclient11.2-
sqlplus-11.2.0.1.0-1.i386.rpm
```

2. Далее требуется настроить некоторые переменные окружения:

```
$ export ORACLE_HOME=/usr/lib/oracle/11.2/client
$ export ORACLE_HOME_LISTNER=/usr/lib/oracle/11.2/client/bin
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH :/usr/lib/oracle/11.2/client/lib
```

```
$ export SQLPATH=/usr/lib/oracle/11.2/client/lib
$ export TNS_ADMIN=/usr/lib/oracle/11.2/client/bin
```

3. Добавьте следующие настройки в файл /etc/odbcinst.ini:

```
[Oracle11g]
Description = Oracle ODBC driver for Oracle 11g
Driver = /usr/lib/oracle/11.2/client/lib/libsqora.so.11.1
```

4. Добавьте DSN в odbc.ini:

```
[ORCLTEST]
Driver     = Oracle 11g ODBC driver
ServerName = <IP-адрес>
Database   = <системный идентификатор базы данных>
DSN        = ORCLTEST
Port       = 1521
```

5. Проверьте возможность подключения:

```
$ isql -v ORCLTEST
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
+---------------------------------------+
```

6. На этом настройку ODBC можно считать законченной.

Конфигурационные файлы unixODBC

Теперь у вас имеется возможность подключаться к наиболее распространенным базам данных. Проверить соединение можно с помощью утилиты isql:

1. Если вы не указали имя пользователя и пароль в файле odbc.ini, его можно передать утилите isql, как показано ниже:

```
$ isql <DSN> <пользователь> <пароль>
```

2. В противном случае достаточно выполнить усеченную команду:

```
$ isql mysql-test
```

3. Если никаких проблем не возникнет, должен появиться следующий вывод:

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

Если на экране появилось сообщение об ошибке, такое как `Data source name not found and no default driver specified` (Имя источника данных не найдено и драйвер по умолчанию не определен), проверьте переменные окружения ODBCINI и ODBCSYSINI – они должны ссылаться на существующий файл `odbc.ini` и его каталог. Например, если вы храните файл `odbc.ini` в каталоге `/usr/local/etc`, эти переменные должны содержать следующие значения:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

4. Если проблема кроется в DSN, выполните следующую команду:

```
$ isql -v <DSN>
```

Ключ `-v` переведет команду в **режим вывода подробных сообщений**, по которым вы сможете определить, где находится ошибка.

Кроме того, помните, что `/etc/odbcinst.ini` является общесистемным файлом, поэтому все ваши драйверы unixODBC должны определяться там.

Компиляция Zabbix с поддержкой ODBC

Если вы подключились к целевой базе данных, подлежащей мониторингу, можно скомпилировать сервер Zabbix с поддержкой ODBC:

Если система Zabbix уже настроена и работает, не выполняйте привычную команду `make install` после компиляции, так как она копирует слишком много файлов, и какие-то файлы в действующей конфигурации окажутся затертыми. В таком случае достаточно скопировать единственный выполняемый файл сервера Zabbix.

1. Введите команду `configure` со всеми параметрами, описанными в *главе 1* «*Развертывание Zabbix*», добавьте ключ `--with-unixodbc` и выполните ее:

```
$ ./configure --enable-server --with-postgresql --with-net-snmp --with-libcurl
--enable-ipv6 --with-openipmi --enable-agent --with-unixodbc
```

2. В выводе команды должны появиться следующие строки:

```
checking for odbc_config... /usr/local/bin/odbc_config
checking for main in -lodbc... yes
checking whether unixodbc is usable... yes
```

3. Это подтвердит, что все необходимые двоичные файлы ODBC найдены и доступны. Когда этап конфигурации завершится, произведите компиляцию:

```
$ make
```

4. Затем создайте резервную копию файла `zabbix_server`, установленного ранее, и скопируйте новую версию на его место.
5. После запуска `zabbix_server` загляните в файл журнала, где должны появиться следующие строки:

```
****** Enabled features ******
SNMP monitoring: YES
```

```
IPMI monitoring: YES
WEB monitoring: YES
Jabber notifications: YES
Ez Texting notifications: YES
ODBC: YES
SSH2 support: YES
IPv6 support: YES
****************************
```

Они означают, что запуск прошел успешно.

Элементы мониторинга базы данных

Теперь пришло время задействовать поддержку ODBC в сервере Zabbix. Для этого создадим элемент типа **Database monitor (Монитор баз данных)**, как показано на рис. 4.5.

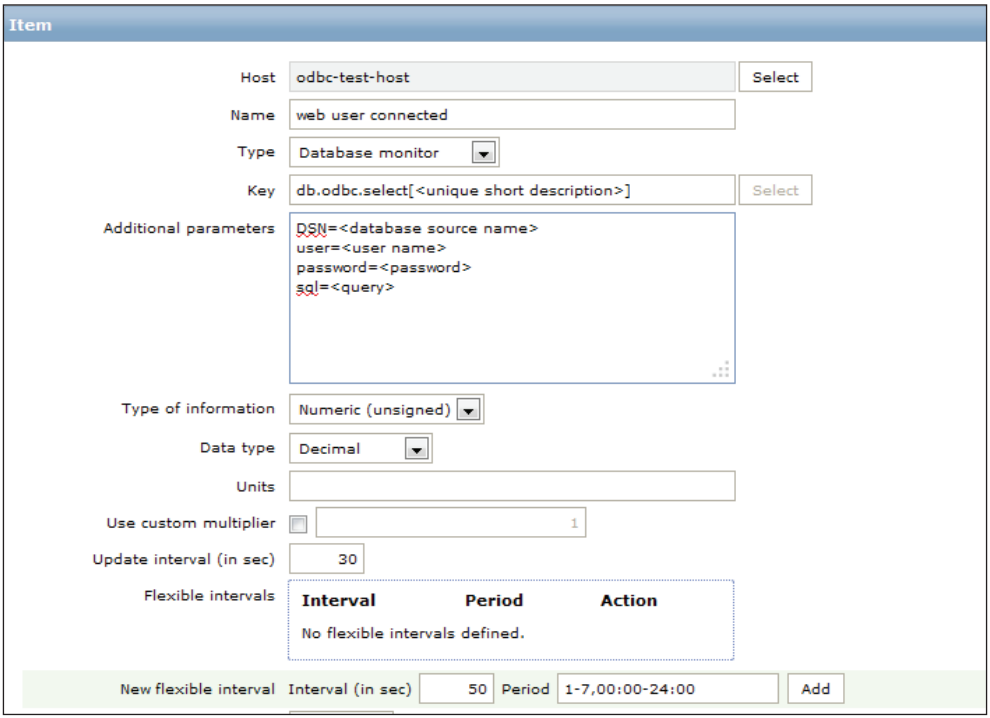


Рис. 4.5 ❖ Создание элемента мониторинга базы данных

Элемент, куда будет сохраняться полученное значение, идентифицируется ключом

```
db.odbc.select[<unique short description>]
```

где <unique short description> – это произвольная уникальная строка на ваш выбор (уникальное краткое описание), например

```
db.odbc.select[web_user_connected_on_myapp]
```

В поле **Additional parameters** (Дополнительные параметры) определите следующие параметры:

```
DSN=<имя источника данных>
user=<имя пользователя>
password=<пароль>
sql=<запрос>
```

Имя источника данных DSN должно быть определено в файле /etc/odbc.ini. Имя пользователя и пароль могут быть указаны в определении DSN или здесь. В последнем параметре следует определить запрос SQL.

Некоторые замечания о запросах ODBC SQL

Ниже описываются некоторые ограничения и замечания, которые следует иметь в виду, создавая запросы SQL:

- SQL-запрос должен начинаться с инструкции SELECT;
- SQL-запрос не должен содержать символов перевода строки;
- SQL-запрос должен возвращать единственное значение;
- если запрос возвращает несколько столбцов, сохранен будет только первый;
- если запрос возвращает несколько строк, сохранен будет только первый столбец первой строки;
- макросы (например, {HOSTNAME}) не разворачиваются;
- команда SQL должна начинаться с символов в нижнем регистре, то есть sql=;
- SQL-запрос должен выполняться достаточно быстро, чтобы не произошло превышение тайм-аута;
- SQL-запрос должен возвращать значение ожидаемого типа; в противном случае элемент не будет поддерживаться.

Как видите, имеются определенные ограничения, которые вы должны принять. В частности, вы не можете вызывать функции, возвращающие больше одного значения. Вы не можете вызывать хранимые процедуры – допускается только извлекать данные (инструкция select). Кроме того, текст запроса должен находиться в одной строке, то есть длинные и сложные запросы будут получаться неудобочитаемыми.

Далее перечисляется несколько замечаний:

- если база данных работает под большой нагрузкой, она может отвечать с некоторой задержкой (задержка может возникать также при попытке соединения);
- для каждого запроса устанавливается новое соединение;
- если соединение с базой данных выполняется по адресу 127.0.0.1, могут возникать некоторые проблемы;
- если в инфраструктуре используются прокси-серверы, они должны быть скомпилированы с поддержкой unixODBC.

Если база данных работает под большой нагрузкой, не создавайте пул соединений, которые создадут ненужную дополнительную нагрузку. Кроме того, в таких случаях установка соединения может занимать больше 5 секунд.

Значение 5 секунд упомянуто выше не случайно – это тайм-аут попытки соединения. В процессе инициализации нужно определить ожидаемую величину тайм-аута, по истечении которого попытку соединиться можно признать неудавшейся.

Этот тайм-аут определяется в файле `src/libs/zbxdbhigh/odbc.c`, в строке 130:

```
SQLSetConnectAttr(pdbh->hdbc, (SQLINTEGER)SQL_LOGIN_TIMEOUT,
    (SQLPOINTER)5, (SQLINTEGER)0);
```

Выражение `(SQLPOINTER)5` присвоит параметру `SQL_LOGIN_TIMEOUT` значение `5` (секунд). Если база данных не успеет ответить в течение 5 секунд, в файле журнала появится следующее сообщение об ошибке:

```
[ODBC 3.51 Driver]Can't connect to MySQL server on 'XXX.XXX.XXX.XXX' (4)] (2003).
```

Попробуйте увеличить значение `SQL_LOGIN_TIMEOUT` до 15 секунд и повторно скомпилировать сервер и прокси:

```
SQLSetConnectAttr(pdbh->hdbc, (SQLINTEGER)SQL_LOGIN_TIMEOUT,
    (SQLPOINTER)15,
    (SQLINTEGER)0);
```

Мониторинг через JMX

В версии Zabbix 2.0 появилась встроенная поддержка мониторинга приложений с использованием механизма JMX. Мониторинг JMX-приложений осуществляется Java-демоном **Zabbix Java gateway**, который действует подобно шлюзу. Когда серверу Zabbix требуется узнать значение конкретного JMX-счетчика, он просто передает запрос шлюзу Java, который, в свою очередь, выполнит необходимую операцию. Все запросы выполняются посредством JMX API.

В настоящее время Zabbix Java gateway находится на ранней стадии своего развития, поэтому, несмотря на богатство функциональных возможностей, все еще может иметь некоторые проблемы.

Единственное требование, предъявляемое этим методом, – контролируемое приложение должно запускаться с поддержкой удаленного мониторинга JMX. Оно не должно реализовать или наследовать какой-то определенный класс, и от него не требуется включать код для обработки запросов Zabbix, потому что эти запросы преобразуются шлюзом в стандартные запросы JMX.

Чтобы запустить приложение с поддержкой удаленного мониторинга JMX, необходимо передать следующие параметры:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=<номер порта>
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

Эти параметры настраивают интерфейс JMX на стороне приложения. Как обычно, нужно определить порт, метод аутентификации и необходимость шифрования.

Это самая простая конфигурация, но, к сожалению, не самая безопасная и защищенная.

Защищенность JMX

Открывая дверь в свое Java-приложение, вы подвергаете его риску нападений извне. Консоль JMX, имеющаяся во многих серверах приложений, позволяет не только получать значения счетчиков, но и выполнять более сложные операции. Фактически с помощью консоли JMX, открытой на сервере приложений, можно развертывать приложения, запускать их и останавливать. Стоит ли упоминать, что взломщик, завладевший доступом к консоли, сможет развернуть свое приложение и запустить его или вызвать проблемы в работе уже действующего приложения. Консоль JMX можно вызывать из самого сервера приложений, используя методы `post` и `get`. Добавлением вредоносного содержимого в раздел `HEAD` веб-страницы злоумышленник сможет легко взломать консоль JMX и получить прямой доступ к вашей инфраструктуре. Скомпрометированный сервер приложений может поставить под угрозу всю вашу сеть, поэтому необходимо предусмотреть превентивные меры, препятствующие этому. Реализация этих мер описывается в следующих шагах.

1. Прежде всего необходимо включить аутентификацию:

```
-Dcom.sun.management.jmxremote.authenticate=true
```

2. Затем определить файл, содержащий ваш пароль:

```
-Dcom.sun.management.jmxremote.password.file=/etc/java-6-penjdk/management/jmxremote.
password
```

Аутентификация паролем удаленных соединений JMX имеет некоторые проблемы с безопасностью. После того как клиент получит дескриптор удаленного соединения из незащищенного реестра RMI (по умолчанию), что часто используется в атаках вида «незаконный посредник» (man-in-the-middle), злоумышленник может запустить подставной реестр RMI на целевом сервере, непосредственно перед оригинальным, и перехватить пароль клиента.

3. Также необходимо указать файл управления доступом:

```
-Dcom.sun.management.jmxremote.access.file=/etc/java-6-penjdk/management/jmxremote.
access
```

4. В файле доступа, например, можно указать следующую информацию:

```
monitorRole readonly
controlRole readwrite
```

5. Файл паролей в этом случае должен содержать следующие строки:

```
monitorRole <пароль для мониторинга>
controlRole <пароль для управления>
```

6. Чтобы избежать утечки паролей, следует включить шифрование SSL:

```
-Dcom.sun.management.jmxremote.ssl=true
```

7. Этот параметр неразрывно связан со следующими:

```
-Djavax.net.ssl.keyStore=<хранилище ключей>
-Djavax.net.ssl.keyStorePassword=<пароль к ключу>
-Djavax.net.ssl.trustStore=<сертификат сервера>
-Djavax.net.ssl.trustStorePassword=<пароль на сервере>
-Dcom.sun.management.jmxremote.ssl.need.client.auth=true
```

Параметр `-D` записывается в файл, запускающий приложение или сервер приложений, который, из-за присутствия в нем паролей, оказывается уязвимым, и его необходимо защитить, сделав недоступным для чтения с привилегиями других учетных записей.

Установка шлюза Zabbix Java gateway

Чтобы скомпилировать шлюз Java gateway, выполните следующие шаги:

1. Сначала установите необходимые пакеты:

```
$ yum install java-devel
```

2. Затем выполните следующую команду:

```
$ ./configure --enable-java
```

3. Она должна вывести следующие строки:

```
Enable Java gateway:   yes
Java gateway details:
  Java compiler:         javac
  Java archiver:         jar
```

4. Они сообщают, что шлюз Java gateway готов. После этого выполните компиляцию и установку:

```
$ make && make install
```

5. Шлюз Zabbix Java gateway будет установлен в каталог:

```
$PREFIX/sbin/zabbix_java
```

6. В этом каталоге будет храниться файл:

```
bin/zabbix-java-gateway-2.0.5.jar
```

7. Ниже перечислены библиотеки, необходимые шлюзу:

```
lib/logback-classic-0.9.27.jar
lib/logback-core-0.9.27.jar
lib/android-json-4.3_r3.1.jar
lib/slf4j-api-1.6.1.jar
```

8. И два конфигурационных файла:

   lib/logback-console.xml
   lib/logback.xml

9. Сценарии запуска и остановки шлюза:

   shutdown.sh
   startup.sh

10. Типичный сценарий, подключаемый сценариями запуска и остановки и содержащий настройки:

    settings.sh

11. Теперь, если у вас настроено соединение через SSL, нужно включить тот же уровень защищенности для Zabbix Java gateway. Для этого добавьте следующий параметр в сценарий запуска:

    ```
    -Djavax.net.ssl.*
    ```

12. Выполнив все вышеперечисленное, добавьте в конфигурационный файл сервера Zabbix следующие строки:

    ```
    JavaGateway=<IP-адрес>
    JavaGatewayPort=10052
    ```

Если предполагается использовать шлюз Java gateway прокси-сервером, добавьте те же параметры `JavaGateway` и `JavaGatewayProperties` в конфигурационный файл прокси.

13. Поскольку по умолчанию Zabbix не запускает ни одного регистратора Java (poller), необходимо настроить их запуск:

    ```
    StartJavaPollers=5
    ```

14. Перезапустите сервер Zabbix или прокси.
15. И запустите шлюз Zabbix Java gateway командой `startup.sh`.

Журнал шлюза, заполняемый записями до уровня важности `info`, находится в `/tmp/zabbix_java.log`.

Шлюз Zabbix Java gateway использует для журналирования библиотеку `logback`, благодаря чему есть возможность изменить уровень важности журналируемых записей или местоположение файла, просто отредактировав файл lib/logback.xml, в частности следующие теги XML:

```
<fileNamePattern>/tmp/zabbix_java.log.%i</
fileNamePattern><root level="info">
```

Здесь также можно изменить все параметры `logrotation`.

Если в работе шлюза Zabbix Java Gateway возникнет какая-то проблема и потребуется провести отладку, запустите шлюз в консольном режиме. Для этого просто закомментируйте переменную `PID_FILE` в сценарии `settings.sh`. Если сценарий

`startup.sh` не найдет параметр `PID_FILE`, он запустит шлюз Java gateway в консоли, а библиотека Logback в этом случае будет использовать конфигурационный файл `lib/logback-console.xml`. Этот конфигурационный файл не только настраивает вывод журнальных записей в консоль, но также изменяет уровень важности до отладочного (`debug`). За дополнительными подробностями о журналировании в Zabbix Java gateway обращайтесь непосредственно к руководству пользователя SLF4J: http://www.slf4j.org/manual.html.

Настройка JMX в Zabbix

Теперь настроим хост JMX для мониторинга с соответствующими элементами JMX. Для этого добавьте в конфигурацию хоста интерфейс JMX и адрес, как показано на рис. 4.6.

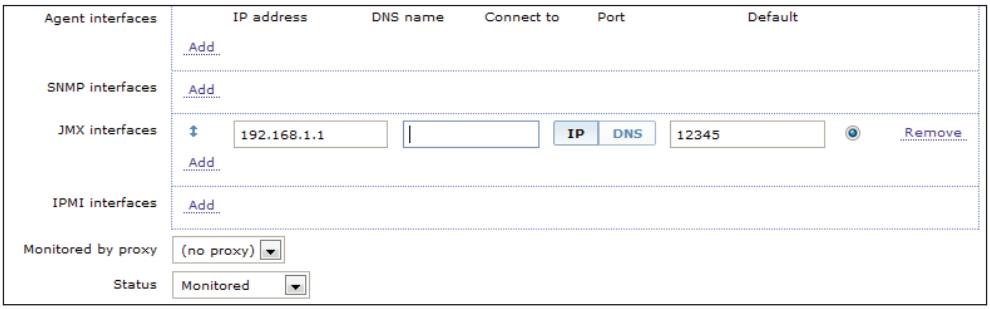


Рис. 4.6 ❖ Добавление интерфейса JMX и адреса

После этого для всех счетчиков JMX, которые вы собираетесь контролировать, необходимо определить элементы типа «**агент JMX**». В определении агента JMX укажите имя пользователя, пароль и строку JMX-запроса. Ключ JMX включает в себя:

- имя объекта `MBean`;
- имя атрибута, то есть имя атрибута объекта `MBean`.

На рис. 4.7 показаны настройки одного из таких элементов.

Поле **Data type** (Тип данных) на рис. 4.7 указывает, что данный элемент может хранить целые числа без знака (такие как 0 и 1) или логические значения (такие как `true` или `false`).

Ключи JMX

Имя объекта MBean в приложении Java – это самая обычная строка. Но с другим компонентом, именем атрибута, дело обстоит сложнее; атрибут может возвращать данные простого или составного типа.

К данным простых типов относятся целые числа и строки. Например, запрос

```
jmx[com.example:Type=Test,weight]
```

вернет вес в виде вещественного числа (с плавающей точкой).

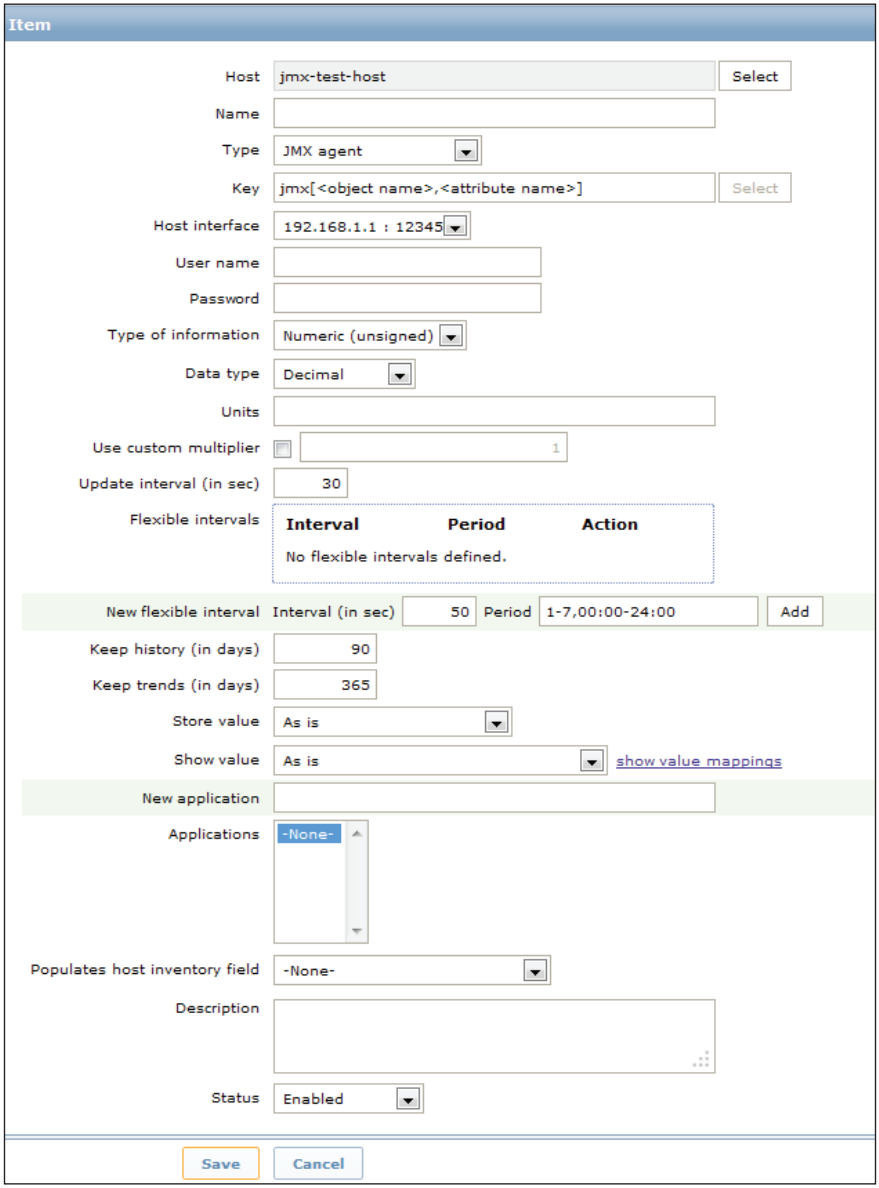


Рис. 4.7 ❖ Настройки элемента типа JMX agent (агент JMX)

Атрибуты, возвращающие составные данные, обрабатывать сложнее, но они поддерживают оператор точки. Например, атрибут `pen` (перо) может иметь два значения, представляющие цвет и остаток чернил, получить которые по отдельности можно с помощью оператора точки, как показано ниже:

```
jmx[com.example:Type=Test,pen.remainink]
jmx[com.example:Type=Test,pen.color]
```

Если у вас имеется атрибут, имя которого включает точку, такое как `all.pen`, эту точку нужно экранировать, как показано ниже:

```
jmx[com.example:Type=Test,all\.pen.color]
```

Если имя атрибута содержит обратный слеш (\), его также нужно экранировать:

```
jmx[com.example:Type=Test,c:\\utility]
```

Если имя объекта или атрибута содержит пробелы или запятые, такое имя следует заключать в дважды повторяющиеся двойные кавычки:

```
jmx[com.example:type=Hello,""c:\\documents and settings""]
```

Некоторые замечания о JMX

К сожалению, поддержка JMX не настолько гибкая и настраиваемая, как хотелось бы; по крайней мере, на момент написания этих строк в JMX оставались нерешенными некоторые проблемы.

Например, по своему личному опыту я знаю, что JBoss, один из самых популярных серверов приложений, нельзя успешно опросить. Конечная точка JMX в настоящее время «жестко зашита» в `JMXItemChecker.java`:

```
service:jmx:rmi:///jndi/rmi://" + conn + ":" + port + "/jmxrmi"
```

Некоторые приложения используют другие конечные точки JMX для управления. JBoss – одно из них. Конечная точка не настраивается ни на уровне хоста, ни на уровне интерфейса, и вы не сможете добавить параметр, определяющий конечную точку в форме настройки хоста.

Однако разработка активно продолжается, и каждый день что-то совершенствуется и улучшается. Бесспорно, что в настоящее время шлюз Zabbix Java gateway требует определенных улучшений. Кроме того, текущая реализация шлюза недостаточно легковесна; если у вас имеется более 100 элементов JMX на хост, шлюз необходимо периодически перезапускать, потому что иначе можно столкнуться, например, с такой ошибкой:

```
failed: another network error, wait for 15 seconds
```

которая сопровождается сообщением:

```
connection restored
```

Существует еще один аспект, на который следует обратить внимание: иногда может так случиться, что на одном хосте будет выполняться сразу несколько экземпляров JVM (виртуальной машины Java). В такой ситуации требуется настроить отдельный порт JMX для каждой группы элементов и определить псевдонимы хостов, по одному для каждого сетевого интерфейса; однако такой сценарий не может быть разрешен низкоуровневыми средствами обнаружения и требует

приложить массу усилий для настройки. Для разработчика инфраструктуры мониторинга на основе Zabbix чрезвычайно важно знать не только проблемы, но и ограничения. Тогда он сможет выбирать между разработкой своего программного обеспечения и использованием открытой альтернативы, попытаться исправить проблему или представить свои предложения разработчикам Zabbix.

Мониторинг через SNMP

Простой протокол сетевого управления (Simple Network Management Protocol, SNMP) в действительности не так прост, как можно было бы заключить из его названия. Он является стандартом де-факто для многих устройств и приложений. Дело не только в его распространенности – часто это единственный способ извлечь контролируемую информацию из сетевого коммутатора, дискового массива, устройства бесперебойного питания и т. д.

Реализовать мониторинг с применением SNMP на самом деле очень просто. На каждом контролируемом хосте или устройстве запускается агент SNMP. Этот агент получает запросы (например, от инструментов командной строки или сервера мониторинга, такого как Zabbix) и возвращает информацию об измеренных параметрах или даже изменяет некоторые предопределенные настройки в ответ на команды, содержащиеся в запросе. Кроме того, агент – не просто пассивный компонент, отвечающий на команды и запросы, но также может посылать оповещения в ловушки SNMP предопределенного хоста, когда выполняются некоторые условия.

Ситуация осложняется, когда дело доходит до количественных определений. В отличие от обычных элементов Zabbix или любой другой системы мониторинга, измеряемые параметры SNMP являются частью большой иерархии, дерева параметров, охватывающего все разнообразие производителей аппаратного программного обеспечения. Это означает, что каждый параметр должен иметь некий уникальный идентификатор, или код. Такой уникальный идентификатор называется OID (Object IDentifier – идентификатор объекта) и идентифицирует объект и его местоположение в иерархии SNMP.

Идентификаторы объектов и их значения составляют фактическое содержимое, передаваемое в сообщения SNMP. Несмотря на эффективность с точки зрения сетевого трафика, идентификаторы OID необходимо преобразовывать в некоторый вид, понятный человеку. Такое преобразование осуществляется с применением распределенной **базы данных управляющей информации (Management Information Base, MIB)**. База данных MIB, по сути, является коллекцией текстовых файлов для различных ветвей дерева OID с текстовыми описаниями и названиями отдельных идентификаторов.

С помощью базы данных MIB можно, например, узнать, что OID **1.3.6.1.2.1.1.3** ссылается на значение времени работы системы с момента последней загрузки (uptime). Его значением является целое число – тысячные доли секунды – и может обозначаться как **sysUpTime**, как показано на следующей диаграмме (рис. 4.8).

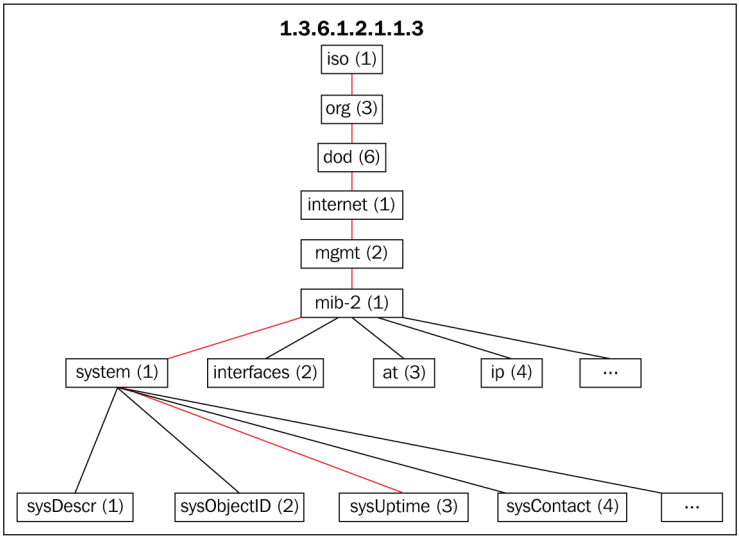


Рис. 4.8 ❖ Дерево SNMP MIB

Как видите, мониторинг SNMP существенно отличается от мониторинга с применением агентов Zabbix – и с точки зрения сетевого протокола, и с точки зрения определений элементов и организации. Тем не менее Zabbix включает средства преобразования идентификаторов SNMP OID в элементы Zabbix – если сервер скомпилирован с поддержкой SNMP, он сможет создавать запросы SNMP и с помощью пары инструментов поддержки обрабатывать ловушки SNMP.

Эта особенность играет важную роль, когда мониторинг устройства возможен только посредством SNMP и нет никакой возможности установить своего агента. Ниже перечислены некоторые причины, оправдывающие выбор SNMP как основного протокола мониторинга в вашей сети и полный отказ от агентов Zabbix.

- Не требуется осуществлять мониторинг большого количества сложных или нестандартных параметров, кроме тех, что уже поддерживаются системной ветвью SNMP OID. Если у вас уже настроен мониторинг SNMP сетевого оборудования и вам достаточно таких простых параметров, как время работы после последней перезагрузки, нагрузка на процессор, объем свободной памяти и др., для их получения проще использовать SNMP, чем устанавливать и настраивать агентов Zabbix. При таком подходе вам не придется волноваться о развертывании агентов и их обновлении – достаточно настроить в сервере Zabbix подключение к удаленным агентам SNMP и получение необходимой информации.
- Протокол SNMP и номера портов хорошо известны практически всем продуктам, и если вам понадобится посылать им данные мониторинга, намного проще положиться на протокол SNMP. Кроме того, порты UDP с номерами 161 и 162 могут быть уже открыты, или сетевого администратора легче бу-

дет убедить разрешить трафик хорошо известного протокола, чем малоизвестного.

- Протокол SNMP версии 3 имеет встроенные механизмы аутентификации и обеспечения безопасности. То есть, в отличие от протокола Zabbix, с которым вы познакомились в *главе 2 «Распределенный мониторинг»*, сообщения SNMPv3 обеспечивают целостность, конфиденциальность и аутентификацию. Несмотря на то что Zabbix поддерживает все три версии протокола SNMP, настоятельно рекомендуется использовать исключительно версию 3, если это возможно, потому что она единственная обладает настоящей защищенностью. Версии 1 и 2 имеют неполноценную систему защиты, основанную на передаче строки в сообщении.
  Но даже при наличии веских причин использовать мониторинг SNMP в инфраструктуре Zabbix иногда предпочтительнее все же не отказываться от агентов Zabbix.
- Агент Zabbix изначально поддерживает измерение некоторых параметров, для получения которых средствами SNMP пришлось бы разрабатывать собственные расширения. Например, если потребуется организовать мониторинг файла журнала, с автоматической ротацией и пропуском устаревших данных, вам достаточно просто определить ключ `logrt[]` для активного элемента Zabbix. Так же просто организуется мониторинг контрольной суммы и размера определенного файла или получение информации из монитора производительности в операционной системе Windows и т. д. Во всех этих случаях агент Zabbix является наиболее очевидным выбором.
- Агент Zabbix поддерживает автоматическое обнаружение вновь появляющихся на хосте ресурсов и отправку информации о них серверу, который, в свою очередь, может автоматически создавать элементы и триггеры и уничтожать их, когда соответствующие ресурсы становятся недоступны. То есть при использовании агента Zabbix можно настроить сервер так, чтобы он создавал соответствующие элементы для каждого процессора на контролируемом хосте смонтированной файловой системы, количества сетевых интерфейсов и т. д. То же самое можно сделать при помощи SNMP, определив низкоуровневые правила обнаружения, но часто для решения этой задачи намного проще использовать агентов Zabbix.

С учетом вышесказанного следует тщательно взвешивать возможности каждого решения и выбрать то, что лучше подходит для вашего окружения. Но в общем случае придерживайтесь следующих рекомендаций: если контролируемые параметры просты, но предъявляются строгие требования к безопасности, используйте SNMPv3; если требуется контролировать комплексные параметры или автоматически обнаруживать ресурсы, и можно обойтись без серьезных мер безопасности (или допускается применение сложных решений обеспечения безопасности, как описывалось в *главе 2 «Распределенный мониторинг»*), используйте агента Zabbix.

А теперь перейдем к обсуждению пары важных аспектов мониторинга с применением протокола SNMP в Zabbix. Сначала мы познакомимся с простыми запросами SNMP, а затем перейдем к ловушкам SNMP.

Запросы SNMP

Элемент мониторинга через SNMP очень прост в настройке. Самое интересное, что, даже несмотря на использование SNMP OID для доступа к контролируемому параметру, все еще необходимо определить уникальное имя и, что самое важное, уникальный ключ элемента. Имейте в виду, что ключи элементов используются во всех выражениях Zabbix, триггерах, вычисляемых элементах, операциях и т. д. Поэтому старайтесь выбирать как можно более короткие и простые ключи, чтобы их проще было распознавать. Например, допустим, что требуется определить контролируемый параметр для входящего сетевого трафика на порт с номером 3 устройства и параметр имеет OID `1.3.6.1.2.1.2.2.1.10.3`. В этом случае можно было бы выбрать ключ `port3.ifInOctects`, как показано на рис. 4.9.

Если элемент SNMP не определен в шаблоне заранее, получить информацию о нем проще всего с помощью утилиты `snmpwalk`, непосредственно послав запрос контролируемому хосту и получив информацию о доступных OID параметров и их типах.

Например, следующая команда вернет полное дерево объектов на устройстве с адресом `10.10.15.19`:

```
$ snmpwalk -v 3 -l AuthPriv -u user -a MD5 -A auth -x DES -X priv -m ALL 10.10.15.19
```

Замените строку `user` именем пользователя для агента SNMP, строку `auth` – паролем пользователя, строку `priv` – приватным паролем, строку `MD5` – названием требуемого протокола аутентификации и строку `DES` – названием приватного протокола, определенного для агента. Не забывайте, что оба пароля должны быть длиннее восьми символов.

Агент SNMP, выполняющийся на хосте, вернет список всех OID, как показано ниже (здесь приведен лишь фрагмент ответа):

```
HOST-RESOURCES-MIB::hrSystemUptime.0 = Timeticks: (8609925) 23:54:59.25HOST-RESOURCES-
MIB::hrSystemDate.0 = STRING: 2013-7-28,9:38:51.0,+2:0
HOST-RESOURCES-MIB::hrSystemInitialLoadDevice.0 = INTEGER: 393216
HOST-RESOURCES-MIB::hrSystemInitialLoadParameters.0 = STRING: "root=/dev/sda8 ro"
HOST-RESOURCES-MIB::hrSystemNumUsers.0 = Gauge32: 2
HOST-RESOURCES-MIB::hrSystemProcesses.0 = Gauge32: 172
HOST-RESOURCES-MIB::hrSystemMaxProcesses.0 = INTEGER: 0
HOST-RESOURCES-MIB::hrMemorySize.0 = INTEGER: 8058172 KBytes
HOST-RESOURCES-MIB::hrStorageDescr.1 = STRING: Physical memory
HOST-RESOURCES-MIB::hrStorageDescr.3 = STRING: Virtual memory
HOST-RESOURCES-MIB::hrStorageDescr.6 = STRING: Memory buffers
HOST-RESOURCES-MIB::hrStorageDescr.7 = STRING: Cached memory
HOST-RESOURCES-MIB::hrStorageDescr.8 = STRING: Shared memory
HOST-RESOURCES-MIB::hrStorageDescr.10 = STRING: Swap space
HOST-RESOURCES-MIB::hrStorageDescr.35 = STRING: /run
```

```
HOST-RESOURCES-MIB::hrStorageDescr.37 = STRING: /dev/shm
HOST-RESOURCES-MIB::hrStorageDescr.39 = STRING: /sys/fs/cgroup
HOST-RESOURCES-MIB::hrStorageDescr.53 = STRING: /tmp
HOST-RESOURCES-MIB::hrStorageDescr.56 = STRING: /boot
```

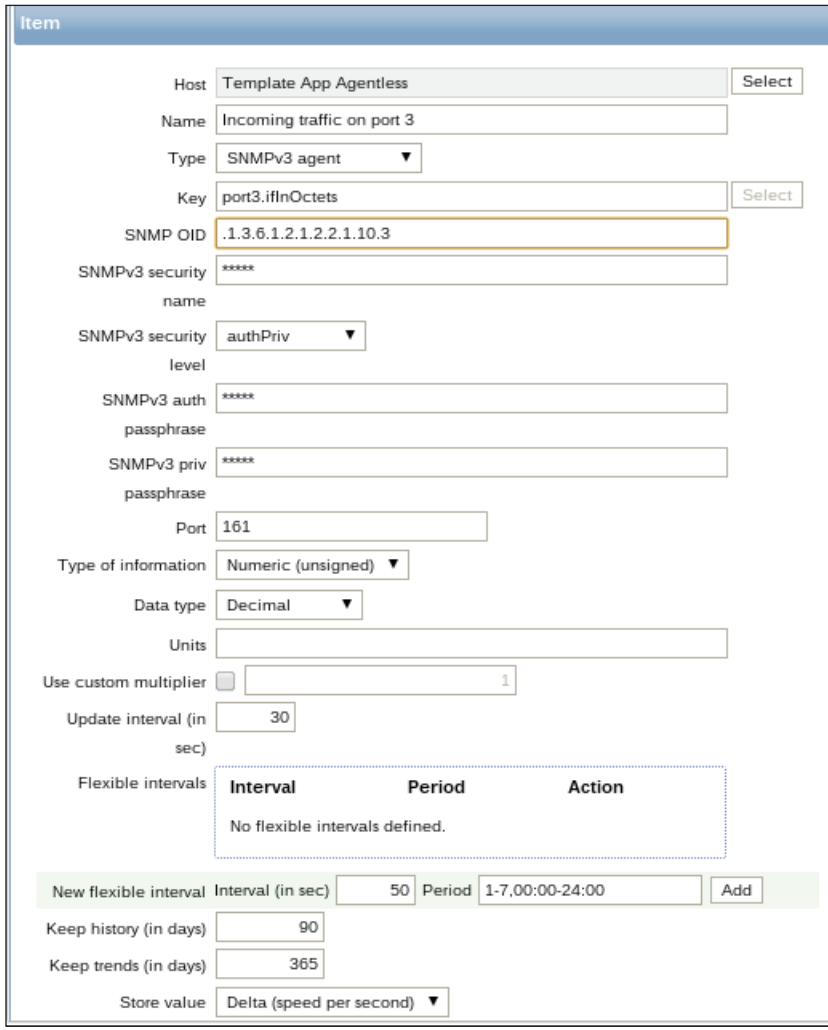


Рис. 4.9 ❖ Настройки контролируемого параметра с OID `1.3.6.1.2.1.2.2.1.10.3`

Допустим, что нас интересует объем памяти в системе. Чтобы получить полный OID для данного параметра, воспользуемся вновь утилитой `snmpwalk` с аргументом `fn` в параметре `-0`. Он предписывает команде `snmpwalk` вывести полные значения OID в числовом формате.

```
$ snmpwalk -v 3 -l AuthPriv -u user -a MD5 -A auth -x DES -X priv -m ALL -O fn 10.10.15.19
...
HOST-RESOURCES-MIB::hrMemorySize.0.1.3.6.1.2.1.25.2.2.0 = INTEGER: 8058172 Kbytes
...
```

Итак, требуемый нам OID: `1.3.6.1.2.1.25.2.2.0`.

Ловушки SNMP

Ловушки SNMP не похожи на элементы Zabbix других типов. В отличие от них, ловушки SNMP не возвращают значения, а генерируют событие. Иными словами, они являются результатом некоторой проверки или вычислений, выполненных агентом SNMP, и отправки в сервер мониторинга сообщения, извещающего об этом. Ловушка SNMP может срабатывать всякий раз, когда подконтрольный хост перезагружается, когда сетевой интерфейс выходит из строя, когда выходит из строя жесткий диск или когда блок бесперебойного питания отключается от сети и переходит на питание сервера от аккумуляторов.

Этот вид информации резко отличается от элементов Zabbix, представляющих результаты измерений, не связанные с какими-либо событиями. С другой стороны, может не быть иных способов узнать о каких-то ситуациях, иначе как посредством ловушки SNMP, потому что они не связаны ни с какими измеряемыми параметрами (например, событие начала процедуры остановки сервера) или потому что устройство может контролироваться только посредством объектов SNMP и ловушек.

Ловушки имеют довольно ограниченное применение в Zabbix, потому что на их основе можно только лишь создавать простые триггеры и затем выводить извещения о событиях (нет особого смысла отображать графики на основе ловушки или создавать вычисляемые элементы). Однако они могут играть важную роль в законченных решениях мониторинга.

Для эффективного управления ловушками SNMP серверу Zabbix требуются два вспомогательных инструмента: демон `snmptrapd`, фактически обрабатывающий соединения с агентами SNMP, и некоторый сценарий, форматирующий каждую ловушку и передающий результат серверу Zabbix для дальнейшей обработки.

Демон snmptrapd

Если вы скомпилировали сервер Zabbix с поддержкой SNMP, значит, у вас уже установлен полный комплект инструментов, в том числе демон SNMP, демон ловушек SNMP и пакет вспомогательных утилит, таких как `snmpwalk` и `snmptrap`.

Если же вы до сих пор не установили поддержку SNMP, выполните следующую команду:

```
# yum install net-snmp net-snmp-utils
```

По аналогии с пакетом демонов, входящих в состав сервера Zabbix и прослушивающих порт TCP с номером 10051 (запросов от агентов, прокси и узлов), `snmptrapd` также является процессом-демоном, прослушивающим порт UDP с номером 162 и принимающим сообщения от удаленных агентов SNMP.

После установки snmptrapd загляните в конфигурационный файл snmptrapd.conf, который обычно находится в каталоге /etc/snmp/. Для настройки snmptrapd требуется, как минимум, определить параметр authCommunity для версий SNMP 1 и 2:

```
authCommunity log public
```

или, для версии SNMPv3, имя пользователя и приватный пароль:

```
createUser -e ENGINEID user MD5 auth DES priv
```

Создайте отдельный параметр createUser для каждого удаленного агента SNMPv3, который должен посылать сообщения в ловушки. Также замените все значения user, auth, priv, MD5 и DES в соответствии с уже выполненными настройками агентов, как описывалось в примечании выше. Не забудьте еще указать правильное значение ENGINEID для каждого агента. Его можно узнать из конфигурации агента.

С этой минимальной конфигурацией snmptrapd ограничится журналированием сообщений в syslog. Эту информацию можно было бы извлекать и пересылать серверу Zabbix, но намного проще переложить обработку ловушек непосредственно на snmptrapd. Хотя демон не имеет собственных инструментов для такой обработки, он может выполнять произвольные команды или приложения, указанные в директиве trapHandle или за счет встроенной поддержки perl. Последний вариант наиболее эффективен, потому что в этом случае демону не придется запускать новый процесс и ждать, пока он завершится. Этот подход особенно рекомендуется при большом количестве ловушек. Просто добавьте следующую строку в snmptrapd.conf:

```
perl do "/usr/local/bin/zabbix_trap_receiver.pl";
```

Сценарий zabbix_trap_receiver можно взять из исходных текстов Zabbix. Он находится в каталоге misc/snmptrap/zabbix_trap_receiver.pl.

После перезапуска демона snmptrapd он будет выполнять сценарий на Perl, указанный вами, для обработки всех ловушек. Как вы уже, вероятно, догадались, ваша работа на этом не заканчивается – в сценарии вам нужно реализовать обработку ловушек и найти способ передать результаты в сервер Zabbix. Оба этих аспекта обсуждаются в следующем разделе.

Обработка ловушек в сценарии на Perl

Сценарий на Perl, включенный в дистрибутив Zabbix, действует как передаточное звено, преобразующее сообщения в формате SNMP в элементы Zabbix. Каждое полученное сообщение форматируется в соответствии с правилами, определенными внутри сценария, а полученный результат выводится в файл журнала. Сервер Zabbix, в свою очередь, следит за указанным файлом журнала и обрабатывает все вновь появляющиеся строки как элементы ловушек SNMP, просто сопоставляя содержимое строки с элементами ловушек, настроенными для соответствующего хоста. Давайте посмотрим, как все это работает, заглянув в сценарий на Perl и исследовав его логику:

```
#!/usr/bin/perl
#
# Zabbix
# Copyright (C) 2001-2013 Zabbix SIA
#
############################################
#### О ПРИЕМНИКЕ СООБЩЕНИЙ SNMP В ZABBIX ####
############################################

# Это встроенный сценарий на Perl, принимающий сообщения SNMP
# и пересылающий данные в сервер.
# Приемник передает полученные сообщения SNMP в сервер Zabbix
# или в прокси, выполняющийся на той же машине, что и сам
# сценарий. Не забудьте настроить сервер/прокси соответственно.
#
# Дополнительную информацию об использовании сценария с
# Net-SNMP можно найти по адресу:
# http://net-snmp.sourceforge.net/wiki/index.php/Tut:Extending_snmpd_using_perl
```

Первый раздел содержит только информацию о лицензировании и краткое описание сценария. Здесь нет ничего примечательного, кроме одного – проверьте ссылку на выполняемый файл `perl` в первой строке и измените ее, если это необходимо. Следующий раздел интереснее, и если вам достаточно простого преобразования сообщений SNMP, это может быть единственный раздел, куда вам придется внести свои изменения:

```
############################################
#### НАСТРОЙКИ ПРИЕМНИКА СООБЩЕНИЙ SNMP ####
############################################

$SNMPTrapperFile = '/tmp/zabbix_traps.tmp';
$DateTimeFormat = '%H:%M:%S %Y/%m/%d';
```

Просто присвойте переменной `$SNMPTrapperFile` путь к файлу журнала, куда предполагается записывать преобразованные сообщения SNMP, и укажите то же значение в параметре `SNMPTrapperFile`, в своем файле `zabbix_server.conf`. Там же, в файле `zabbix_server.conf`, присвойте параметру `StartSNMPTrapper` значение `1`, чтобы сервер приступил к мониторингу данного файла.

Значение в переменной `$DateTimeFormat` должно отражать фактический формат принимаемых сообщений SNMP. Значение, присвоенное по умолчанию, обычно подходит для большинства ситуаций, но вам следует убедиться в этом и исправить при необходимости.

Следующий раздел содержит фактическую логику сценария. Обратите внимание на большой объем логики в подпрограмме `zabbix_receiver`. Эта подпрограмма вызывается в конце сценария и заслуживает, чтобы мы рассмотрели ее во всех деталях:

```
################################
#### ПРИЕМНИК СООБЩЕНИЙ SNMP ####
```

```
################################
use Fcntl qw(O_WRONLY O_APPEND O_CREAT);
use POSIX qw(strftime);
sub zabbix_receiver
{
    my (%pdu_info) = %{$_[0]};
    my (@varbinds) = @{$_[1]};
```

Демон snmptrapd запускает сценарий и передает ему вновь принятое сообщение. Сценарий, в свою очередь, вызывает эту подпрограмму, которая сразу же записывает сообщение в два списка – первый аргумент добавляется в хэш %pdu_info, а второй в массив @varbinds:

```
    # открыть файл для вывода
    unless (sysopen(OUTPUT_FILE, $SNMPTrapperFile,
                    O_WRONLY|O_APPEND|O_CREAT, 0666))
    {
        print STDERR "Cannot open [$SNMPTrapperFile]:$!\n";
        return NETSNMPTRAPD_HANDLER_FAIL;
    }
```

Здесь сценарий открывает файл для вывода или автоматически завершается, если попытка не увенчалась успехом. На следующем шаге извлекается имя хоста (или IP-адрес) агента, отправившего сообщение. Эта информация хранится в хэше %pdu_info, объявленном выше:

```
    # извлечь имя хоста
    my $hostname = $pdu_info{'receivedfrom'} || 'unknown';
    if ($hostname ne 'unknown') {
        $hostname =~ /\[(.*?)\].*/;
        $hostname = $1 || 'unknown';
    }
```

Теперь все готово к сборке фактического сообщения SNMP. Первая часть вывода используется сервером Zabbix для распознавания новой ловушки (поиском по строке ZBXTRAP и определением хоста, для которого определена ловушка). Имейте в виду, что IP-адрес (или имя хоста), извлеченный здесь, должен совпадать с адресом SNMP в конфигурационном файле хоста, указанном в веб-интерфейсе Zabbix. Это значение должно быть установлено, даже если оно идентично IP-адресу (имени заданного хоста). После идентификации хоста сервер Zabbix отбрасывает эту часть сообщения:

```
    # вывод заголовка сообщения
    # в начало первой строки должна быть помещена метка времени
    # (ее можно опустить)
    # первая строка должна включать заголовок
    # "ZBXTRAP [адрес IP/DNS]"
    #     * адрес IP/DNS используется для поиска
    #       соответствующего элемента ловушки SNMP
    #     * этот заголовок удаляется в ходе обработки
```

```
    #       (не записывается в значение элемента)

    printf OUTPUT_FILE "%s ZBXTRAP %s\n",
    strftime($DateTimeFormat, localtime), $hostname;
```

Вслед за заголовком сценарий выводит остальную часть сообщения в том виде, в каком она была принята от агента SNMP:

```
    # вывести содержимое pdu_info
    print OUTPUT_FILE "PDU INFO:\n";
    foreach my $key(keys(%pdu_info))
    {
        printf OUTPUT_FILE " %-30s %s\n", $key,
        $pdu_info{$key};
    }
```

Предыдущий фрагмент выполняет обход содержимого хэша `%pdu_info` и выводит пары ключ/значение:

```
    # вывести связанные переменные:
    print OUTPUT_FILE "VARBINDS:\n";
    foreach my $x (@varbinds)
    {
        printf OUTPUT_FILE " %-30s type=%-2d value=%s\n",
            $x->[0], $x->[2], $x->[1];
    }
    close (OUTPUT_FILE);
    return NETSNMPTRAPD_HANDLER_OK;
}
```

Следующий цикл выполняет инструкцию, `printf OUTPUT_FILE " %-30s type=%-2d value=%s\n", $x->[0], $x->[2], $x->[1];`, которая выводит содержимое массива `@varbinds`. Этот массив содержит фактические значения, посланные внутри сообщения. После этого файл журнала закрывается, и подпрограмма возвращает код успешного завершения:

```
NetSNMP::TrapReceiver::register("all", \&zabbix_receiver) or
    die "failed to register Zabbix SNMP trap receiver\n";
print STDOUT "Loaded Zabbix SNMP trap receiver\n";
```

Последние несколько строк в сценарии устанавливают подпрограмму `zabbix_receiver` в качестве фактического обработчика и выводят сообщение, извещающее об успешном выполнении настроек. После того как обработчик ловушек начнет заполнять файл журнала `zabbix_traps.log`, вам нужно определить соответствующие элементы Zabbix.

Как вы уже видели, первая часть строки, что выводится в файл журнала, используется сервером Zabbix для поиска хоста. Вторая часть добавляется в соответствующие исторические значения элементов. Это означает, что если желаете запустить элемент ловушки для данного хоста, его следует настроить с ключом `snmptrap["coldStart"]`, как показано на рис. 4.10.

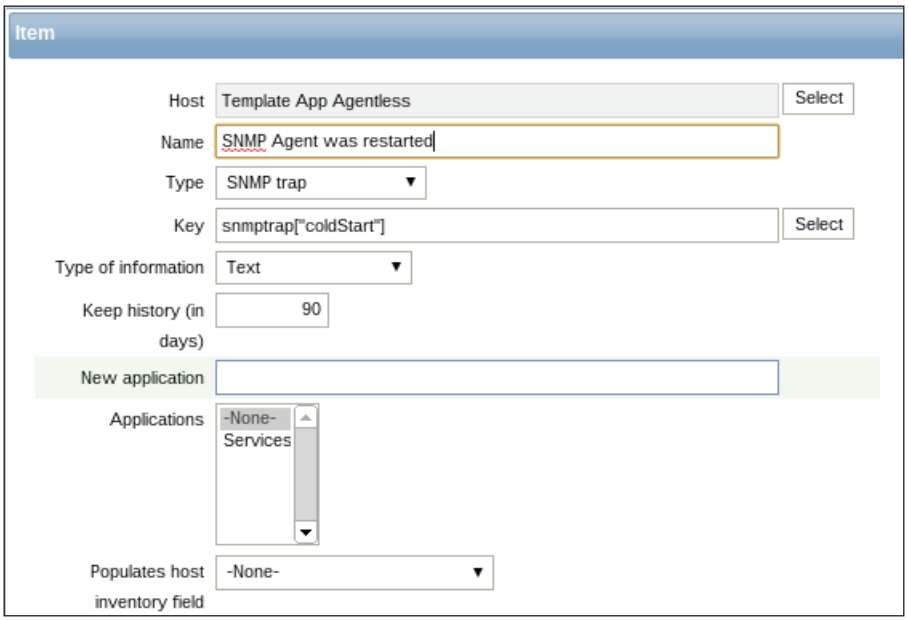


Рис. 4.10 ❖ Параметры настройки элемента ловушки SNMP

С этого момента содержимое ловушки будет сохраняться в истории элемента.

Мониторинг через SSH

Мониторинг через SSH выполняется сервером Zabbix вообще без участия какого-либо агента, что очень удобно в некоторых ситуациях. Данная возможность особенно ценна тем, что позволяет выполнять удаленные команды на устройствах, не поддерживающих возможности установки агента Zabbix. Она может использоваться везде, где по каким-то причинам нет возможности установить агента Zabbix, например:

- для мониторинга устройств, не позволяющих устанавливать дополнительное программное обеспечение;
- для мониторинга устройств с необычной или закрытой операционной системой.

Чтобы получить возможность осуществлять мониторинг через SSH, сервер Zabbix должен быть скомпилирован с поддержкой SSH2; минимально необходимая библиотека – `libssh2` версии 1.0.0.

Мониторинг через SSH поддерживает два вида аутентификации:

- с именем пользователя и паролем;
- с файлом ключа.

Чтобы использовать форму аутентификации с именем пользователя и паролем, не требуется никаких особых настроек; достаточно скомпилировать сервер Zabbix с поддержкой SSH2.

Настройка SSH-аутентификации с ключом

Чтобы использовать форму аутентификации с файлом ключа, необходимо добавить настройки в файл `zabbix_server.conf`; в частности следует изменить следующий параметр:

```
# SSHKeyLocation=
```

Раскомментируйте эту строку и укажите в ней каталог, где хранятся публичный и приватный ключи, например:

```
SSHKeyLocation=/home/zabbixsvr/.ssh
```

После этого перезапустите сервер Zabbix, выполнив следующую команду от имени root:

```
$ service zabbix-server restart
```

Затем можно создать новую пару ключей SSH, выполнив следующую команду от имени root:

```
$ sudo -u zabbix ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/zabbix/.ssh/id_rsa):
Created directory '/home/zabbix/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/zabbix/.ssh/id_rsa.
Your public key has been saved in /home/zabbix/.ssh/id_rsa.pub.
The key fingerprint is:
a9:30:a9:ce:c6:22:82:1d:df:33:41:aa:df:f3:e4:de zabbix@localhost.
localdomain
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                 |
|     ..  .       |
|    +o  S        |
|  ...o..         |
|.o.+ ....        |
|=o= ..=o .       |
|ooo.. .*+ E      |
+-----------------+
```

После этого на удаленном устройстве создайте учетную запись с ограниченными привилегиями, чтобы не подвергать систему ненужному риску, и скопируй-

те полученные ключи. В следующем примере предполагается, что на удаленном устройстве была создана учетная запись `zabbix_mon`:

```
$ sudo -u zabbix ssh-copy-id zabbix_mon@<remote-host-ip>
```

Теперь можно проверить выполненные настройки, попытавшись соединиться с устройством:

```
$ sudo -u zabbix ssh zabbix_mon@<remote-host-ip>
```

Если все было сделано правильно, откроется сеанс связи с удаленным устройством.

Наконец, можно определить элемент для получения вывода команды `uname -a` и сохранения версии ядра. Настройки этого элемента показаны на рис. 4.11.

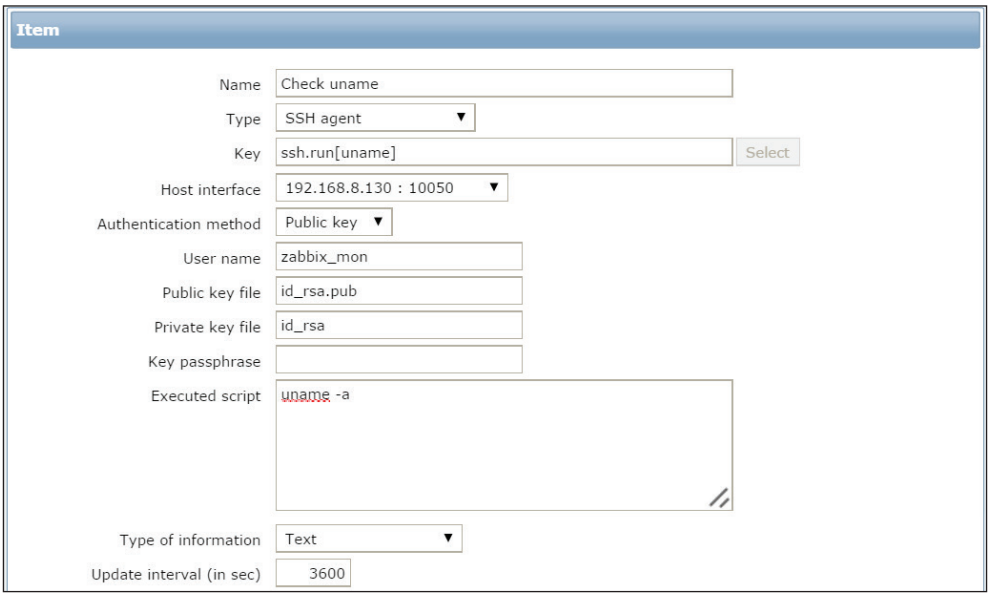


Рис. 4.11 ❖ Настройки пробного элемента для получения версии ядра

В заключение необходимо упомянуть о некоторых ограничениях. Во-первых, библиотека `libssh2` может ограничивать вывод 32 КБ. Во-вторых, всегда желательно использовать полные пути ко всем командам. В-третьих, протокол SSH может вносить свою задержку и замедлять весь процесс мониторинга. Все эти замечания справедливы и для мониторинга через Telnet, который имеет свои отрицательные стороны. Протокол Telnet не поддерживает шифрования и является небезопасным. Кроме того, как вы уже знаете, Telnet поддерживает аутентификацию только с использованием имени пользователя и пароля. Если вы собираетесь осуществлять мониторинг через Telnet, создайте учетную запись, обладающую правами только для чтения.

Мониторинг через IPMI

Не так давно появилась возможность контролировать состояние и доступность устройств с использованием интерфейса IPMI. Главное требование в этом случае: ваше устройство должно поддерживать **интеллектуальный интерфейс управления платформой (Intelligent Platform Management Interface, IPMI)**. Интерфейс IPMI является аппаратной спецификацией, то есть он не связан с конкретной операционной системой или BIOS. Одной из интересных особенностей интерфейса IPMI является его доступность, даже когда система не запущена. Это возможно благодаря наличию в каждом устройстве с поддержкой IPMI отдельного аппаратного устройства с низким энергопотреблением, не зависящего от аппаратного или программного окружения. В настоящее время интерфейс IPMI поддерживается большинством производителей серверов и реализуется в виде карт управления: HP ILO, IBM RSA, Sun SSP, DELL RDAC и т. д.

Подробное описание особенностей работы интерфейса IMPI, разработанного компанией Intel, можно найти в электронной документации по адресу: http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-specifications.html.

Очевидно, что для мониторинга через IPMI необходимо скомпилировать сервер Zabbix с его поддержкой, с флагом `--with-openipmi`, как описывалось в *главе 1* «*Развертывание Zabbix*».

Для организации диалога со всеми компонентами устройства интерфейс IPMI использует протокол запрос/ответ, но самое интересное, что, кроме параметров работы компонентов или доступа к хранимому журналу системных событий, есть возможность читать данные со всех датчиков, установленных в устройстве.

Первые шаги с IPMI

Для начала необходимо установить все требуемые пакеты, например выполнив следующую команду от имени root:

```
$ yum install ipmitool OpenIPMI OpenIPMI-libs
```

После этого можно попробовать получить замеры температуры, как показано ниже:

```
$ ipmitool sdr list | grep Temp
Ambient Temp | 23 degrees C | ok
CPU 1 Temp | 45 degrees C | ok
CPU 2 Temp | disabled     | ns
CPU 3 Temp | disabled     | ns
CPU 4 Temp | disabled     | ns
```

Обратите внимание на три строки со словом `disabled` (отключено). Они указывают, что соответствующие слоты для процессоров пусты. Как видите, интерфейс IPMI позволяет очень быстро получать все внутренние параметры. Теперь интересно будет посмотреть все поддерживаемые параметры для IPMI-идентификатора

CPU 1 Temp. Обратите внимание, что из-за наличия пробелов в идентификаторе его необходимо заключить в двойные кавычки:

```
$ ipmitool event "CPU 1 Temp" list
Finding sensor CPU 1 Temp... ok
Sensor States:
  lnr : Lower Non-Recoverable
  lcr : Lower Critical
  lnc : Lower Non-Critical
  unc : Upper Non-Critical
  ucr : Upper Critical
  unr : Upper Non-Recoverable
```

Это все возможные состояния CPU 1 Temp. Интерфейс IPMI – это простой протокол, поддерживающий только операции чтения, но с его помощью все еще можно имитировать ошибки и настраивать параметры. Попробуем смоделировать нижний порог температуры, чтобы посмотреть, как это делается. Следующая команда установит нижний порог, равный –128 °C:

```
$ ipmitool event "CPU 1 Temp" "lnc : Lower Non-Critical"
Finding sensor CPU 1 Temp... ok
0 | Pre-Init Time-stamp | Temperature CPU 1 Temp | Lower Non-critica l | going low |
Reading -128 < Threshold -128 degrees C
```

Теперь можно быстро убедиться, что его превышение было зарегистрировано в журнале событий:

```
$ ipmitool sel list | tail -1
1c0 | 11/19/2008 | 21:38:22 | Temperature #0x98 | Lower Non-critical going low
```

Это один из наглядных примеров, показывающих, почему желательно создать учетную запись IPMI, обладающую только возможностью чтения. Учетная запись администратора IPMI позволяет сбросить управляемый контроллер, перезагрузить систему, изменить список загрузочных устройств и т. д.

Настройка учетных записей IPMI

Настроить учетную запись IPMI можно двумя основными способами:

- посредством самого интерфейса управления (RDAC, ILO, RS и др.);
- с помощью инструментов операционной системы и OpenIPMI.

Для начала желательно изменить пароль root по умолчанию:

```
$ ipmitool user set password 2 <новый_пароль>
```

Эта команда изменит пароль по умолчанию пользователя root (с идентификатором 2).

Затем нужно создать учетную запись для пользователя Zabbix, который сможет запрашивать данные, но не сможет перезапускать сервер или изменять его настройки.

Далее описывается создание учетной записи для пользователя zabbix (с идентификатором 3). Но перед этим проверьте, нет ли уже пользователя с идентификато-

ром 3 в вашей системе. Сначала определим имя пользователя, выполнив следующую команду от имени root:

```
$ ipmitool user set name 3 zabbix
```

Затем зададим пароль:

```
$ ipmitool user set password 3
Password for user 3:
Password for user 3:
```

Теперь определим привилегии пользователя zabbix:

```
$ ipmitool channel setaccess 1 3 link=on ipmi=on callin=on privilege=2
```

Активируем учетную запись:

```
$ ipmitool user enable 3
```

и проверим:

```
$ ipmitool channel getaccess 1 3
Maximum User IDs : 15
Enabled User IDs : 2
User ID : 3
User Name : zabbix
Fixed Name : No
Access Available : call-in / callback
Link Authentication : enabled
IPMI Messaging : enabled
Privilege Level : USER
```

Мы только что создали учетную запись для пользователя zabbix и назначили ей уровень привилегий USER. Но в данный момент эта учетная запись недоступна из сети; чтобы разрешить сетевой доступ, нужно активировать аутентификацию MD5 для доступа LAN к группе USER:

```
$ ipmitool lan set 1 auth USER MD5
```

Проверим:

```
$ ipmitool lan print 1
Set in Progress    : Set Complete
Auth Type Support  : NONE MD5 PASSWORD
Auth Type Enable   : Callback :
                   : User     : MD5
                   : Operator :
                   : Admin    : MD5
                   : OEM      :
```

Теперь можно выполнять запросы удаленно, с сервера Zabbix, используя следующую команду:

```
$ ipmitool -U Zabbix -H <IP-адрес_хоста_IPMI> -I lanplus sdr list | grep Temp
Ambient Temp | 23 degrees C | ok
CPU 1 Temp | 45 degrees C   | ok
CPU 2 Temp | disabled    | ns
CPU 3 Temp | disabled    | ns
CPU 4 Temp | disabled    | ns
```

Теперь все готово к использованию сервера Zabbix для извлечения элементов IPMI.

Настройка элементов IPMI в Zabbix

Самое сложное в организации получения данных через интерфейс IPMI – это настройка, которую мы только что выполнили. Настройка элементов Zabbix не вызывает никаких сложностей. Прежде всего необходимо раскомментировать следующую строку в файле `zabbix_server.conf`:

```
# StartIPMIPollers=0
```

Подставьте значение, наиболее подходящее для количества опрашиваемых интерфейсов IPMI. В любом случае это значение не является критически важным; самое важное – включить регистратор (poller) IPMI Zabbix, который отключен по умолчанию, например:

```
StartIPMIPollers=5
```

Теперь перезапустите сервер Zabbix, выполнив от имени root следующую команду:

```
$ service zabbix-server restart
```

Теперь перейдем в веб-интерфейс и добавим элементы IPMI.

Первый шаг – настройка параметров IPMI на уровне хоста. Для этого перейдите на вкладку **Configuration ⇨ Host** (Настройка ⇨ Узлы сети) и добавьте интерфейс IPMI и порт, как показано на рис. 4.12.

Рис. 4.12 ❖ Настройка интерфейса IPMI на уровне хоста

Затем перейдите на вкладку **IPMI**, где находятся другие параметры настройки.

На вкладке **IPMI** в поле **Authentication algorithm** (Алгоритм аутентификации) выберите значение **MD5** и, согласно настройкам, выполненным ранее, в поле **Privilege level** (Уровень привилегий) выберите значение **User**. В поле **Username** (Имя пользователя) укажите имя `zabbix`, а в поле **Password** (Пароль) – пароль, который вы настроили в процессе настройки учетной записи IPMI, как показано на рис. 4.13.

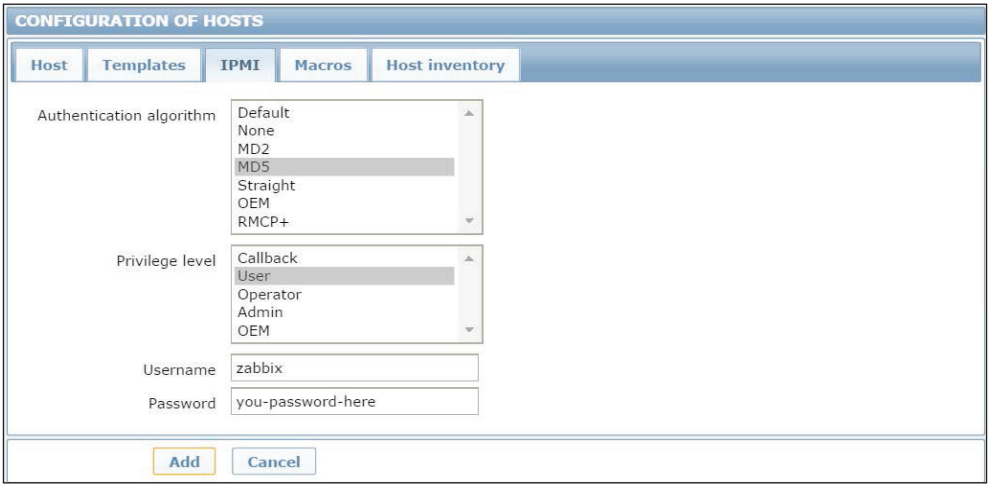


Рис. 4.13 ❖ Настройки дополнительных параметров интерфейса IPMI

Теперь добавьте элемент типа **IPMI agent** (Агент IPMI). По аналогии с предыдущим примером, дадим элементу имя **CPU 1 Temp**, а в поле **Type** (Тип) выберем тип **Numeric (float)** (Числовой (с плавающей точкой)). Результат должен выглядеть, как показано на рис. 4.14.

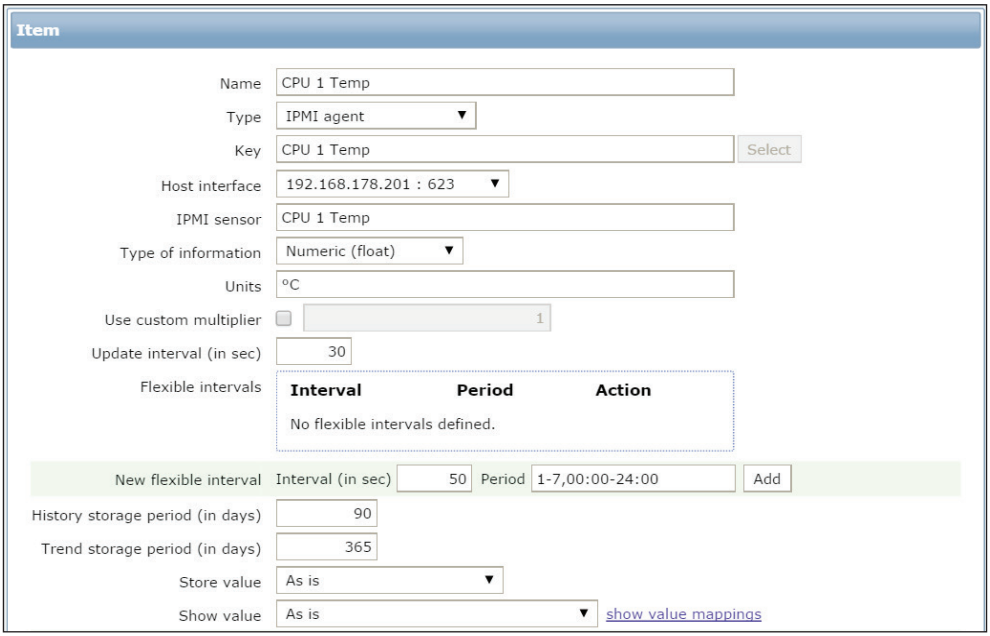


Рис. 4.14 ❖ Настройки элемента IPMI

Настройка элементов Zabbix для получения данных через интерфейс IPMI – очень простой процесс. Если вы используете другую версию OpenIPMI, знайте, что версия OpenIPMI 2.0.7 имеет определенные проблемы и Zabbix плохо работает с ней. Для нормальной работы необходима версия 2.0.14 или выше. Некоторые устройства, такие как сетевые карты с температурными датчиками, могут иметь собственный интерфейс IPMI. В этом случае доступ к таким устройствам осуществляется с использованием того же IP-адреса. Еще одно важное замечание об интерфейсе IPMI: имена дискретных датчиков в OpenIPMI 2.0.16, 2.0.17, 2.0.18 и 2.0.19 могут отличаться. Поэтому желательно проверять соответствие имен используемой версии OpenIPMI.

Мониторинг веб-страниц

В наши дни веб-приложения получили повсеместное распространение. Некоторые веб-сайты или коллекции веб-страниц обычно представляют собой законченный продукт или службу со сложной структурой, включающей базы данных, серверы приложений, веб-серверы, прокси-серверы, балансировщики нагрузки, приборы и многое другое. Планируя мониторинг таких образований, желательно шагнуть на один шаг дальше и предусмотреть проверку получающихся веб-страниц, помимо внутренних ресурсов, скрытых за их фасадом. Предупреждения и уведомления, получаемые при этом, сами по себе имеют ограниченную пользу, так как недоступность веб-страницы определенно является критически важным событием, и они едва ли способны дать более или менее полное представление о возникшей проблеме, в отсутствие настроенных параметров и триггеров мониторинга внутренних ресурсов. С другой стороны, иногда важно иметь данные, помогающие оценить производительность веб-сайта. Для предотвращения возможных проблем и планирования обновлений аппаратного и программного обеспечения часто полезно настроить создание отчетов об уровне обслуживания.

Одним из больших достоинств средств веб-мониторинга в Zabbix является идея сценариев. Она позволяет определить единственный сценарий, выполняющий последовательность простых шагов, опирающихся друг на друга и использующих общие данные. Кроме того, каждый сценарий включает автоматическое создание значимых элементов и графиков как на уровне сценария (общая производительность), так и на уровне каждого шага (локальная производительность). Это позволяет не только проверить работу отдельной страницы, но и сымитировать целый сеанс, в котором каждый компонент веб-приложения вносит свой вклад в получение окончательного результата. Такой сценарий может получиться очень большим и сложным и потребовать создания большого количества элементов, которые непросто группировать и проверять. Именно поэтому в системе Zabbix предусмотрены отдельный интерфейс и своя вкладка для настройки веб-мониторинга.

Для веб-мониторинга сервер Zabbix должен быть скомпилирован с поддержкой cURL (`libcurl`). За дополнительными подробностями обращайтесь к *главе 1* «*Развертывание Zabbix*».

Веб-сценарии поддерживают HTTP/HTTPS, BASIC, NTLM, аутентификацию на основе форм, cookies, отправку форм, проверку содержимого страниц и кодов HTTP-ответов.

Но, несмотря на всю свою мощь, сценарии имеют некоторые ограничения.

Во-первых, веб-сценарии не поддерживают JavaScript, поэтому с их помощью нельзя сымитировать полноценный AJAX-сеанс, доступный для простого пользователя. Это также означает, что автоматическая загрузка содержимого страниц посредством AJAX не будет выполняться сценарием.

Кроме того, чтобы послать форму, требуется заранее знать названия полей и какие данные они должны содержать. Если формы генерируются динамически (как часто бывает в приложениях на ASP.NET, сохраняющих в формах информацию о сеансе), вы не сможете использовать их в последующих шагах.

Эти ограничения могут кому-то показаться незначительными, но они приобретают особую важность, когда требуется организовать мониторинг сайта, в значительной степени полагающегося на операции, выполняющиеся на стороне клиента (JavaScript и пр.), или на динамически генерируемые значения и поля форм. Беда в том, что количество таких веб-приложений увеличивается с каждым днем.

Но даже с этими ограничениями механизм веб-мониторинга в Zabbix оказывается очень полезным и действенным инструментом, который может пригодиться вам, особенно если ваш программный конвейер генерирует большое количество веб-страниц.

Аутентификация для мониторинга веб-страниц

Чтобы создать веб-сценарий, откройте вкладку **Configuration** ⇨ **Host** (Настройка ⇨ Узлы сети) и щелкните на ссылке **Create scenario** (Создать сценарий). В результате откроется форма, изображенная на рис. 4.15.

В этой форме можно определить параметры, такие как **Name** (Имя), **Application** (Приложение) и **Update interval** (Интервал обновления), последний из которых определяет частоту выполнения сценария. Также можно определить агента пользователя (поле **Agent** (Агент)) и количество попыток (поле **Retries** (Попыток)). После выбора агента пользователя сервер Zabbix будет действовать как указанный браузер, представляясь им. Выбирая значение для поля **Retries** (Попыток), важно помнить, что сервер Zabbix не будет повторять тот или иной шаг в случае получения неверного ответа или строки, не совпадающей с ожидаемой.

Недавно появился еще один важный раздел: **Headers** (Заголовки). Здесь можно указать заголовки HTTP, которые будут посылаться сервером Zabbix в запросах.

Начиная с версии Zabbix 2.4 поддерживаются нестандартные заголовки. В этом поле можно использовать макрос `HOST.*`, а также макросы, определенные пользователем.

Механизм веб-мониторинга поддерживает три способа аутентификации; их можно увидеть на вкладке **Authentication** (Аутентификация), как показано на рис. 4.16.

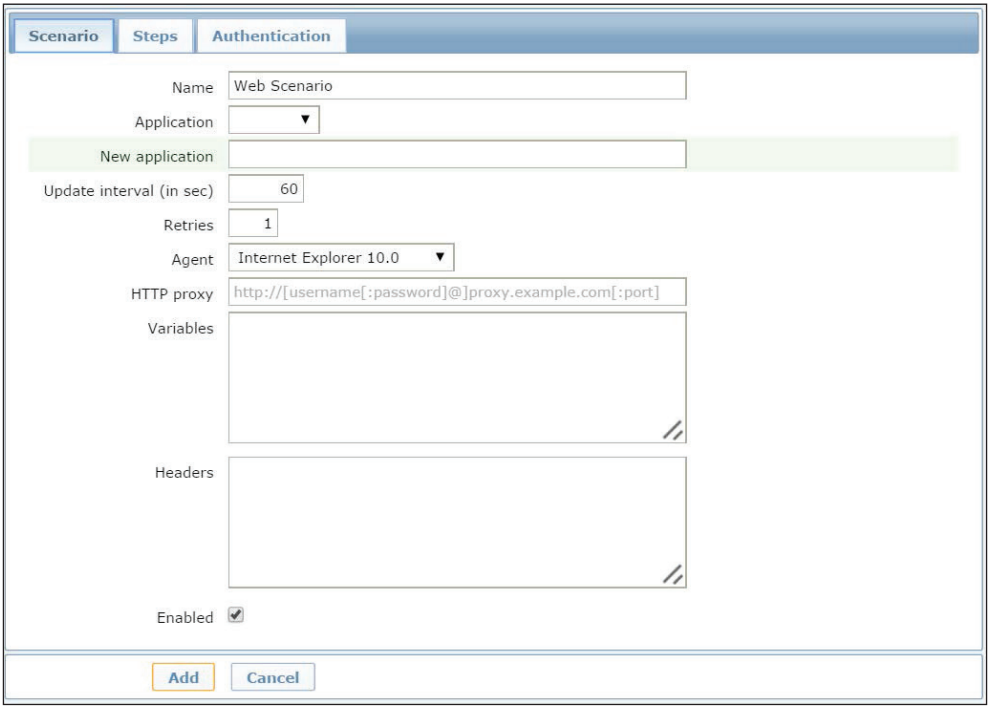


Рис. 4.15 ❖ Форма создания веб-сценария

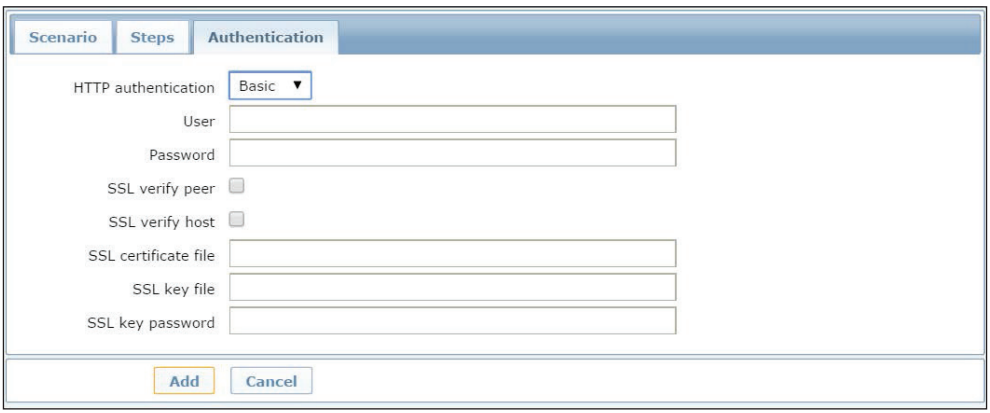


Рис. 4.16 ❖ Вкладка **Authentication** (Аутентификация)

В их числе: Basic, NTLM и на основе форм. Первые два очень просты и определяются на уровне сценария. После выбора аутентификации NTLM в форме с настройками появятся два дополнительных поля для ввода имени пользователя

и пароля. Начиная с версии Zabbix 2.2 в полях с именем пользователя и паролем можно использовать пользовательские макросы. На этой же вкладке есть возможность включить проверку SSL. Флажок **SSL verify peer** (Проверка SSL-узла) управляет проверкой сертификата веб-сервера, а флажок SSL verify host (Проверка SSL-хоста) – управляет проверкой совпадения полей `Common Name` и `Subject Alternate Name` в сертификате веб-сервера. Флажок **SSL certificate file** (Файл SSL-сертификата) определяет имя файла SSL-сертификата для аутентификации клиента; здесь следует указать файл сертификата в формате PEM. Если сертификат в формате PEM содержит приватный ключ, поля **SSL key file** (Файл SSL-ключа) и **SSL key password** (Пароль к SSL-ключу) можно оставить пустыми.

Местоположение сертификата настраивается в главном конфигурационном файле, `zabbix_server.conf`. Для этого существуют три конфигурационных параметра: `SSLCertLocation`, `SSLKeyLocation` и `SSLCALocation`.

Оба поля, **SSL certificate file** (Файл SSL-сертификата) и **SSL key file** (Файл SSL-ключа), поддерживают макрос `HOST.*`.

Вернемся к аутентификации. Нам нужна аутентификация на основе формы, дающая клиенту (в данном случае серверу Zabbix) возможность сохранить сеансовые cookies, которые сообщаются клиенту, когда он посылает форму с данными аутентификации. Определяя сценарий, вам придется выделить один шаг для аутентификации. Чтобы узнать, какие поля должны отправляться с формой, загляните в разметку HTML-страницы с этой формой. В следующем примере демонстрируется форма аутентификации Zabbix. Каждая форма имеет свои отличительные черты, но общая структура похожа (здесь показана только сама форма):

```
<form action="index.php" method="post">
  <input type="hidden" name="request" class="input hidden"
         value="" />
  <!-- Форма входа -->
  <div>Username</div>
  <input type="text" id="name" name="name" />
  <div>Password</div>
  <input type="password" id="password" name="password" />
  <input type="checkbox" id="autologin" name="autologin"
         value="1" checked="checked" />
  <input type="submit" class="input" name="enter"
         id="enter" value="Sign in" />
</form>
```

Обращайте внимание на теги `input` и их атрибуты `name`, потому что именно они представляют поля, которые должны отправляться на сервер вместе с формой. В данном случае поле с именем пользователя имеет имя `name`, поле ввода пароля имеет имя `password` и, наконец, поле `submit` с именем `enter` и значением `Sign in`.

Теперь можно приступать к созданию сценария; мы будем определять сценарий в веб-интерфейсе, изображенном на рис. 4.17.

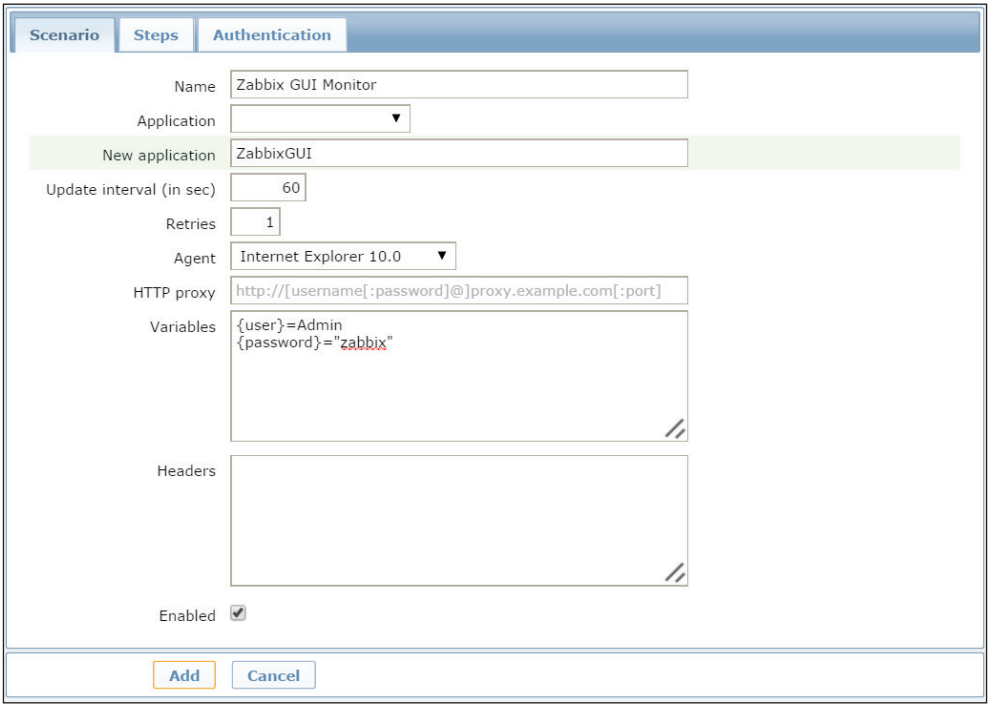


Рис. 4.17 ❖ Веб-интерфейс для создания сценария

Как видите, в поле **Variables** (Переменные) определены две переменные, которые будут использоваться в следующих шагах и в шаге аутентификации. Эта удобная возможность позволяет определять переменные, доступные во всем сценарии.

Далее определим шаг аутентификации, для чего добавим в сценарий новый шаг, как показано на рис. 4.18.

Обратите внимание на использование предопределенных переменных `{user}` и `{password}`. В поле **Required string** (Требуемая строка) введем строку `Connected`, которая появляется в нижнем колонтитуле после соединения, и, конечно, в поле **Required status** (Требуемый код состояния) введем `200`. В этом примере мы определили новую переменную, представляющую токен аутентификации. Она понадобится нам для выполнения процедуры выхода и будет заполнена принятыми данными. С этого момента все запросы на получение страниц или отправку форм будут выполняться в контексте аутентифицированного сеанса, если, конечно, аутентификация увенчалась успехом.

Начиная с версии Zabbix 2.4 все шаги поддерживают переадресацию. Если установлен флажок **Follow redirects** (Следовать перенаправлениям), Zabbix будет использовать cURL-параметр `CURLOPT_FOLLOWLOCATION` (http://curl.haxx.se/libcurl/c/CURLOPT_FOLLOWLOCATION.html). Также если установлен флажок **Retrieve only headers** (Получать только заголовки), сервер будет получать

только заголовки страниц, установив cURL-параметр `CURLOPT_NOBODY` (http://curl.haxx.se/libcurl/c/CURLOPT_NOBODY.html).

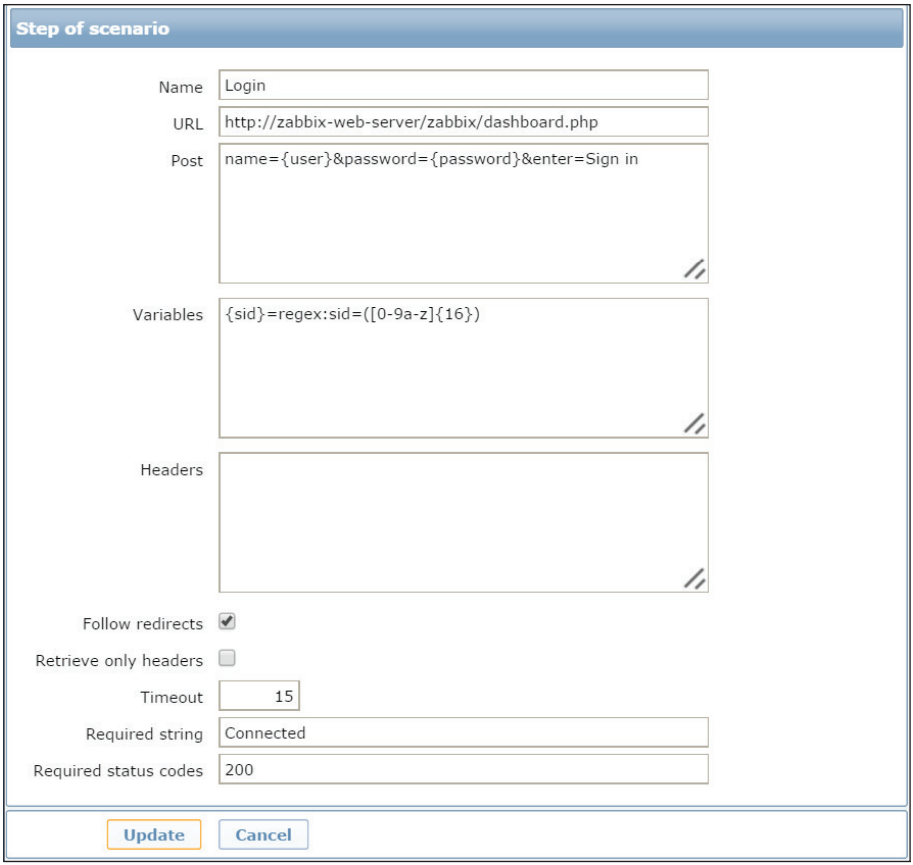


Рис. 4.18 ❖ Шаг аутентификации

Завершение сеанса

При веб-мониторинге многие допускают ошибку, заботясь об аутентификации в начале сценария и забывая завершить сеанс в конце. Если не завершить сеанс, некоторые системы продолжают хранить информацию об аутентифицированных пользователях и открытых сеансах, что может вызвать множество проблем.

Активные сеансы обычно продолжаются от нескольких минут до нескольких дней. Если вы следите за количеством аутентифицированных пользователей и предельное время хранения открытых сеансов достаточно велико, каждая операция аутентификации, выполняемая сценарием, будет увеличивать количество активных пользователей, передаваемых в элемент данных. Если сценарий не закрывает сеанс в конце, вы, как минимум, получите ненадежные, недостоверные результаты

измерений, показывающие наличие большого числа активных сеансов, которые в действительности являются следами, оставленными системой мониторинга.

В худшем случае система аутентификации может не справиться с обработкой слишком большого количества сеансов и сделать невозможной работу всей инфраструктуры. Уверяю вас, что это не гипотетические предположения, а реальные факты, с которыми мне приходилось сталкиваться.

Как бы то ни было, вы будете совершенно правы, добавив операцию завершения сеанса (выхода) в каждый свой веб-сценарий. Таким способом вы предотвратите непредвиденные проблемы, которые мог бы вызвать мониторинг, и заодно проверите правильную работу процедур завершения сеанса. Шаг, завершающий сеанс, обычно очень прост, так как для выхода часто достаточно послать единственный запрос `GET` по требуемому адресу URL. Для выхода из веб-интерфейса Zabbix, например, в сценарии можно определить заключительный шаг, изображенный на рис. 4.19.

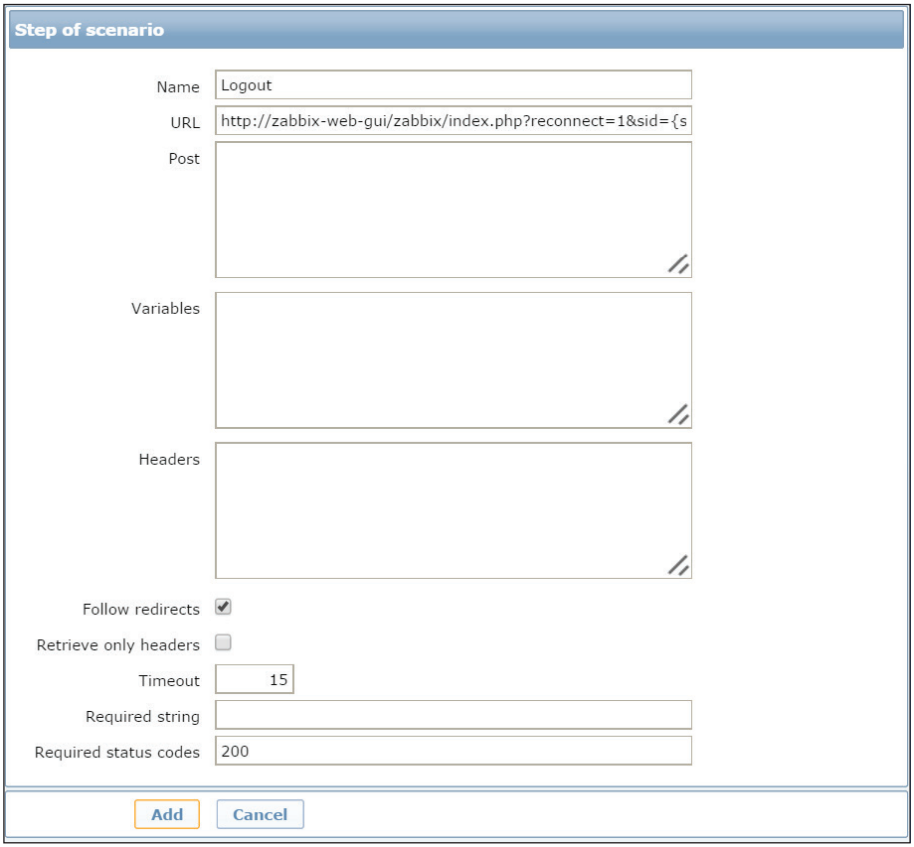


Рис. 4.19 ❖ Заключительный шаг, закрывающий сеанс

После добавления этого заключительного шага ваш сценарий будет выглядеть, как показано на рис. 4.20.

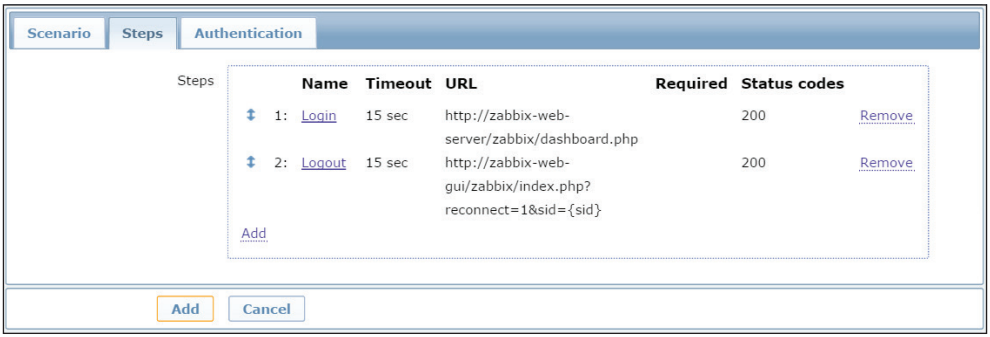


Рис. 4.20 ❖ Веб-сценарий после добавления заключительного шага

Обратите внимание на переменную `{sid}`, использованную в строке с параметрами запроса, осуществляющего выход. Также отметьте, что в данном примере использован адрес URL `zabbix-web-gui` – его следует заменить URL вашего веб-сервера.

Кроме того, имейте в виду, что открытие каждого нового сеанса приводит к расходованию небольшого количества ресурсов, дискового пространства или памяти. Если из-за слишком частых проверок будет создано большое количество сеансов, это может привести к заметному снижению производительности веб-сайта. Поэтому:

- включайте все обязательные шаги в сценарий;
- избегайте создания большого количества похожих сценариев для выполнения простых проверок;
- всегда определяйте шаг завершения сеанса;
- обдуманно выбирайте частоту выполнения сценариев, чтобы не оказывать существенного влияния на подконтрольную систему.

Также вы должны знать, что нет никакой возможности организовать пропуск шагов, включенных в веб-сценарий. Они выполняются в том порядке, в каком определены. Кроме того, если потребуется более подробное журналирование, его можно увеличить прямо во время работы системы, выполнив команду:

```
$ zabbix_server -R log_level_increase="http poller"
```

И последний совет: имейте в виду, что исторические данные в нашем примере хранятся 30 дней и тренды – 90 дней.

Агрегированные и вычисляемые элементы

Все типы элементов, описанные к настоящему моменту, можно рассматривать как способ получения фактических значений, каждое из которых представляет единственную точку на графике. В действительности в данной главе больше внимания

уделялось настройке извлечения разных видов данных в Zabbix, чем фактическому сбору информации. Это объясняется, с одной стороны, тем, что правильная настройка оказывает существенное влияние на эффективность сбора данных и мониторинг, а с другой – тем, что полезность того или иного измерения значительно изменяется в разных системах и конфигурациях, в зависимости от имеющихся потребностей.

Когда дело доходит до агрегированных и вычисляемых элементов, ситуация становится еще интереснее. Элементы этих двух типов никак не связаны с получением данных, но опираются на результаты измерений и помогают глубже проникнуть в суть данных, собранных в вашем окружении.

Это одна из областей, где философия Zabbix разделения измерений и логики обработки дает свои плоды, потому что реализовать нечто подобное иными способами было бы намного сложнее.

Агрегированные и вычисляемые элементы обладают следующими особенностями.

- Элементы обоих типов не выполняют никаких проверок (с применением агентов, внешних сценариев, SNMP, JMX или любого другого программного обеспечения), а напрямую обращаются к базе данных Zabbix с целью обработки имеющейся информации.
- Из-за особенностей организации данных в Zabbix они должны быть связаны с определенным хостом, но это очень слабая связь, если сравнивать с обычными элементами. В действительности агрегированный элемент можно связать с любым хостом, независимо от его назначения, тем не менее, чтобы упростить ссылки на эти элементы в будущем, для них специально определяется один или несколько простых выделенных хостов.
- Агрегированные и вычисляемые элементы могут представлять только данные числовых типов – нет никакой возможности реализовать получение суммы или среднего нескольких текстовых фрагментов.

Агрегированные элементы

Агрегированные элементы – простейший из двух типов, обсуждаемых здесь, – могут выполнять разного рода вычисления с определенным элементом, созданным для всех хостов в группе. Для каждого хоста в данной группе агрегированный элемент получает данные из указанного элемента и ко всем собранным значениям применяет функцию группировки. В результате получается агрегированное значение, действительное на момент выполнения вычислений.

Чтобы создать агрегированный элемент, сначала нужно выбрать группу хостов, а затем элемент, общий для всех хостов в группе, образующий основу для вычислений. Например, допустим, что требуется получить некоторую информацию об активных сеансах в веб-приложениях, действующих под управлением серверов Tomcat. В этом случае группа могла бы иметь имя, например `Tomcat Servers`, а ключ интересующего элемента мог бы иметь вид: `jmx["Catalina:type=Manager,path=/,host=localhost",activeSessions]`.

Далее необходимо решить, как получать данные с каждого хоста, потому что вы не ограничены единственным последним значением и можете производить различные предопределенные вычисления. Кроме функции `last`, которая действительно возвращает последнее значение из истории элемента, все остальные функции (перечисленные в табл. 4.2) принимают дополнительный параметр, определяющий период времени.

Таблица 4.2 ❖ Агрегатные функции

Функция	Описание
`avg`	Возвращает среднее значение за указанный период
`sum`	Возвращает сумму всех значений за указанный период
`min`	Возвращает минимальное значение за указанный период
`max`	Возвращает максимальное значение за указанный период
`last`	Возвращает последнее записанное значение
`count`	Возвращает количество значений за указанный период

Теперь у вас имеется несколько значений, которые требуется сгруппировать. В табл. 4.3 перечислены имеющиеся в вашем распоряжении функции группировки.

Таблица 4.3 ❖ Функции группировки

Функция	Описание
`grpavg`	Возвращает среднее по всем собранным значениям
`grpsum`	Возвращает сумму всех собранных значений
`grpmin`	Возвращает минимальное из всех собранных значений
`grpmax`	Возвращает максимальное из всех собранных значений

Теперь, после знакомства со всеми компонентами агрегированных элементов, можно определить ключ, используя следующий синтаксис:

```
функция_группировки["Группа хостов","Ключ элемента",агрегатная_функция,период_времени]
```

Группу хостов можно определить локально, в определении агрегированного элемента. Если потребуется сгруппировать данные с разных хостов, не входящих в одну группу, и нет возможности создать группу хостов специально для этой цели, вместо имени группы можно указать список хостов, например: `["HostA, HostB, HostC"]`.

Продолжая пример, допустим, что требуется определить среднее количество активных сеансов на сервере приложений Tomcat за каждый час. В этом случае ключ элемента мог бы выглядеть так:

```
grpavg["Tomcat servers",
  "jmx["Catalina:type=Manager,path=/,
  host=localhost",activeSessions]", avg, 3600]
```

Аналогично можно было бы использовать значение `1h` или `60m` в качестве периода времени, если нет желания использовать секунды.

Используя ту же группу и элемент, можно каждые 5 минут получать общее количество активных сеансов на всех серверах:

```
grpsum["Tomcat servers",
  "jmx["Catalina:type=Manager,path=/,host=localhost",maxActive]",
  last, 0]
```

Несмотря на свою простоту, агрегированные элементы помогают получить полезную информацию, которую трудно было бы извлечь, не имея коллекции простых измерений в базе данных.

Вычисляемые элементы

Элементы этого типа основываются на понятии функций элементов, представленном в предыдущих абзацах, поднятом на новый уровень. В отличие от агрегированных элементов, вычисляемые элементы не связаны с группами хостов, и, что особенно важно, вычисления могут выполняться сразу с несколькими разными элементами (то есть с разными ключами). В вычисляемых элементах можно использовать любые функции, доступные в определениях триггеров для любых элементов в базе данных, и объединять разные вычисления с помощью арифметических операций. Ключ вычисляемого элемента не используется для определения фактического источника данных, но он все еще должен быть уникальным, чтобы на него можно было ссылаться в триггерах, графиках и операциях. Фактическое определение вычисляемого элемента находится в поле `formula`, и, как нетрудно догадаться, оно должно быть максимально простым.

Продолжая пример с серверами Tomcat, можно было бы определить вычисляемый элемент, возвращающий полную пропускную способность приложения для данного сервера:

```
last(jmx["Catalina:type=GlobalRequestProcessor,name=http-
  8080",bytesReceived]) +
  last(jmx["Catalina:type=GlobalRequestProcessor,name=http-8080",
  bytesSent]) +
  last(jmx["Catalina:type=GlobalRequestProcessor,name=http-8443",
  bytesReceived]) +
  last(jmx["Catalina:type=GlobalRequestProcessor,name=http-8443",
  bytesSent]) +
  last(jmx["Catalina:type=GlobalRequestProcessor,name=jk-8009",
  bytesReceived]) +
  last(jmx["Catalina:type=GlobalRequestProcessor,name=jk-8009",
  bytesSent])
```

Также можно было бы получить отношение числа активных сеансов к максимально допустимому числу сеансов, чтобы позднее можно было определить триггер, основанный на процентном отношении, а не на абсолютной величине:

```
100*last(jmx["Catalina:type=Manager,path=/,
  host=localhost",activeSessions]) /
```

```
last(jmx["Catalina:type=Manager,path=/,host=localhost",
maxActiveSessions])
```

Как уже отмечалось выше, вычисляемые элементы не ограничиваются единственным хостом.

Следующий пример демонстрирует, как каждые 3 минуты получать среднее количество запросов к базе данных, выполняемых в течение одного сеанса:

```
avg(DBServer:mysql.status[Questions], 180) /
  avg(Tomcatserver:Catalina:type=Manager,path=/,host=localhost",
  activeSessions], 180)
```

Единственное ограничение вычисляемых элементов связано с отсутствием функций группировки, доступных в агрегированных элементах. Поэтому, несмотря на более широкие возможности вычисляемых элементов, вам не удастся полностью избавиться от агрегированных элементов в конфигурациях, когда необходимо применение функций группировки.

Но даже с этим ограничением возможности вычисляемых элементов потрясают воображение. Вместе с агрегированными элементами они являются идеальным инструментом мониторинга производительности групп хостов, таких как кластеры, или сопоставления разных параметров на разных хостах, влияющих на общую производительность службы.

Для каких бы целей не использовались вычисляемые и агрегированные элементы – для анализа производительности и планирования мощностей, или как основа для сложных интеллектуальных триггеров, или для того и другого, их разумное использование поможет получить максимум от инфраструктуры Zabbix.

В заключение

В этой главе мы рассмотрели разные аспекты, связанные с определением элементов в Zabbix. На данный момент вы знаете, чем отличаются элементы Zabbix от объектов мониторинга в других продуктах и почему идея сбора простых данных вместо мониторинга событий выглядит намного предпочтительнее. Вы также знаете теперь, как осуществляется передача данных мониторинга, каковы связанные с этим достоинства и недостатки и как влиять на это движение, чтобы привести его в соответствие с вашими потребностями и потребностями окружения. У вас не должно вызывать трудностей использование других источников данных, помимо стандартного агента Zabbix, таких как базы данных, агенты SNMP, агенты IPMI, веб-страницы, консоли JMX и т. д.

Наконец, вы наверняка поняли, какие широкие возможности сулят агрегированные и вычисляемые элементы.

В следующей главе вы узнаете, как использовать богатые возможности представления и визуализации данных в виде графиков, карт сетей и комплексных экранов.

Глава 5

Визуализация данных

Zabbix – очень гибкая система мониторинга. После настройки она готова выполнять самую тяжелую работу и помогать анализировать огромные объемы данных, представляя их в самых разных видах. Следующий наш шаг – настройка графического отображения данных, интерполяция и выявление взаимосвязей между измерениями. Особенно примечательными являются возможность отображения измерений разного типа в одной системе координат, анализ нагрузки с применением шаблонов, выявление служб и оборудования, выходящих из строя чаще других, и поиск взаимосвязей между параметрами связанных служб.

Помимо стандартных графиков, Zabbix предоставляет возможность создавать собственные графики и добавлять их в шаблоны, позволяя тем самым легко и просто распространять графики на все серверы. Такие нестандартные (а также стандартные и простые) графики можно собирать в комплексные экраны. Такой экран может содержать разные виды информации – простые графики, нестандартные графики, другие экраны, текстовые сообщения, обзоры состояний триггеров и др.

В этой главе мы рассмотрим следующие темы:

- создание собственных графиков;
- создание и использование карт сетей и вложенных карт;
- создание динамических экранов;
- создание и настройка слайд-шоу для отображения на больших мониторах;
- создание отчета об уровне обслуживания (Service Level Agreement, SLA).

В качестве практического примера представьте большой вычислительный центр с многоуровневой системой поддержки; обычно на первом уровне требуется иметь общий обзор происходящего в вычислительном центре, второй уровень может быть реализован как аналог первого, но разбитый по типам услуг, таким как базы данных, серверы приложений и т. д. Администратору баз данных (второй уровень поддержки), к примеру, необходимо иметь всю информацию, относящуюся к работе баз данных, а специалисту по серверам приложений – всю информацию о работе Java, плюс еще несколько стандартных параметров, таких как нагрузка на процессор и объем свободной памяти. Система Zabbix отвечает на все эти требования возможностью создания карт, экранов и слайд-шоу.

После создания графиков и извлечения всех необходимых параметров и сообщений легко можно создать экраны, включающие, например, все графики, свя-

занные с базами данных, плюс стандартные параметры. Экраны можно включать в слайды и на каждом уровне поддержки демонстрировать свои группы экранов в виде слайд-шоу, позволяя специалистам получать подробную и качественную информацию о происходящем.

Поддержка вычислительного центра является, пожалуй, наиболее сложным в реализации слайд-шоу, но в этой главе вы увидите, как легко они создаются. Как только будут готовы все составляющие (простые и нестандартные графики, триггеры и другие компоненты), вы сможете многократно использовать их в разных представлениях. В большинстве слайдов, например, все важные параметры, такие как нагрузка на процессор, объем свободной или занятой памяти и объем сетевого трафика, отображаются в виде графиков. После создания собственных графиков вы сможете использовать их во множестве динамических элементов. Zabbix поддерживает еще одну интересную возможность – создание динамических **карт**. Карта – это графическое представление инфраструктуры сети. Все эти особенности будут описаны в данной главе.

Когда вы наконец подготовите все необходимое для создания экрана, вам также потребуется учесть целевую аудиторию, их навыки, знания и потребности. В простейшем случае вы должны знать, какую информацию следует сообщить в виде графиков.

Графики – мощное средство передачи информации и гибкий инструмент, который можно использовать для усиления эффекта и получения качественной обзорной картинки о службе или инфраструктуре. Эта глава поможет вам с пользой применить все графические элементы Zabbix.

Графики

Графики в Zabbix можно разбить на две категории – простые и нестандартные графики. Обе они рассматриваются в следующем разделе.

Простые графики

Простые графики в Zabbix действительно очень просты, потому что не требуют прилагать больших усилий для их настройки. Просто перейдите на вкладку **Monitoring ⇨ Latest data** (Мониторинг ⇨ Последние данные) и щелкните на ссылке **Graph** (График) напротив выбранного элемента – Zabbix отобразит график истории, как показано на рис. 5.1.

Совершенно понятно, что отображать в виде графиков можно только числовые элементы. Для всех остальных видов информации, таких как текст, графики не поддерживаются, и рядом с такими элементами вместо ссылки **Graph** (График) вы увидите ссылку **Show the history** (Показать историю).

Простые графики не требуют настройки, но вы можете влиять на их отображение.

В верхней части графика находится ползунок выбора периода времени. Если увеличить этот период, на графике появятся агрегированные данные. Если вы-

бран короткий период и просмотр самых последних данных, на графике отображается единственная линия. Если увеличить период до такой степени, что для его отображения потребуется обращение к трендам, на графике появятся три линии. Это отражает связь между историей и трендами; пока для отображения графика достаточно данных из таблицы истории, на графике отображается единственная линия. Но как только потребуется извлечь данные из трендов, появятся три линии, как показано на рис. 5.2.

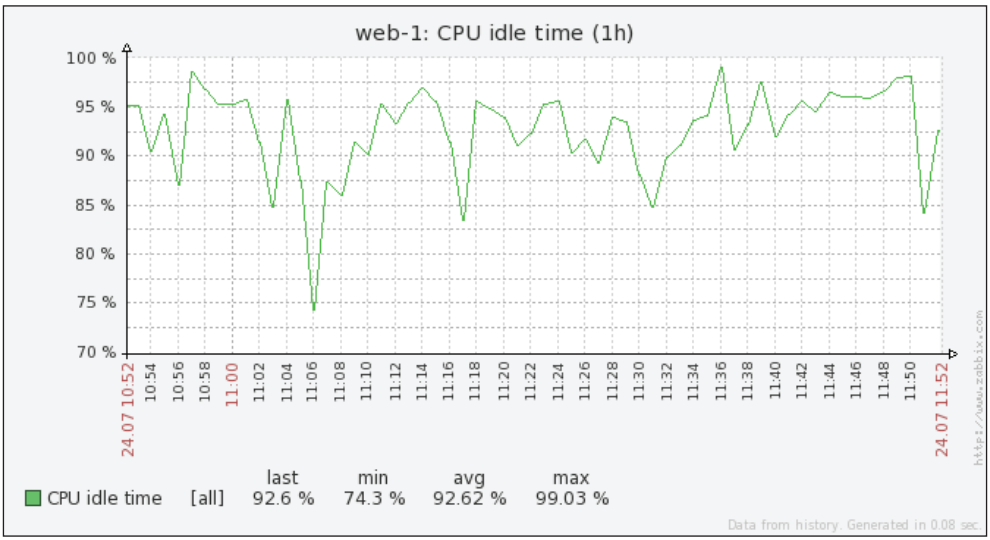


Рис. 5.1 ❖ Простой график

На рис. 5.2 можно видеть три линии, определяющие желтую область. Эта область ограничивается линиями, соответствующими минимальному и максимальному значениям, а зеленая линия представляет среднее. Более полное обсуждение трендов/истории приводится в *главе 1* «*Развертывание Zabbix*».

Продолжительность хранения элемента в истории определяется самим элементом в поле **Keep history (in days)** (Период хранения истории (в днях)), а продолжительность хранения в трендах – в поле **Keep trends (in days)** (Период хранения динамики изменений (в днях)).

На рис. 5.3 можно видеть, как иногда среднее значение мало зависит от минимального и максимального значений. В частности, обратите внимание, что в **12:00** среднее значение почти не изменилось. В этот момент наблюдается существенное падение времени простоя процессора (светло-зеленая линия), которое не оказало существенного влияния на среднее значение (зеленая линия), скорее всего, из-за кратковременности пика нагрузки. Но на графике это падение хорошо видно, потому что Zabbix хранит минимальное и максимальное значения.

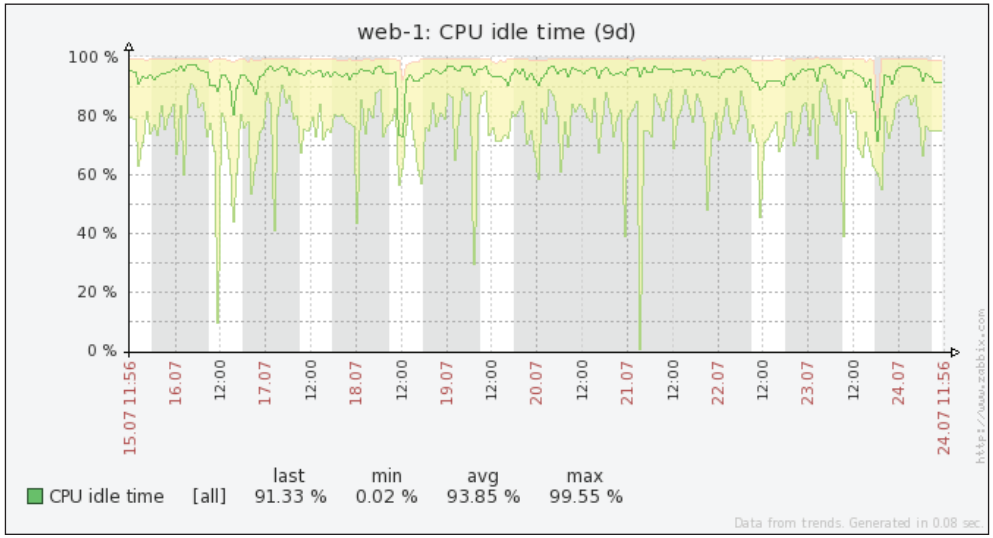


Рис. 5.2 ❖ Как только потребуется извлечь данные из трендов, появятся три линии

Рабочее время (рабочие часы или дни) отображается на графиках белым фоном, а нерабочее время – серым (если в веб-интерфейсе используется оригинальный шаблон). Рабочее время не выделяется, если период, отображаемый графиком, превышает 3 месяца. Это хорошо видно на рис. 5.3.

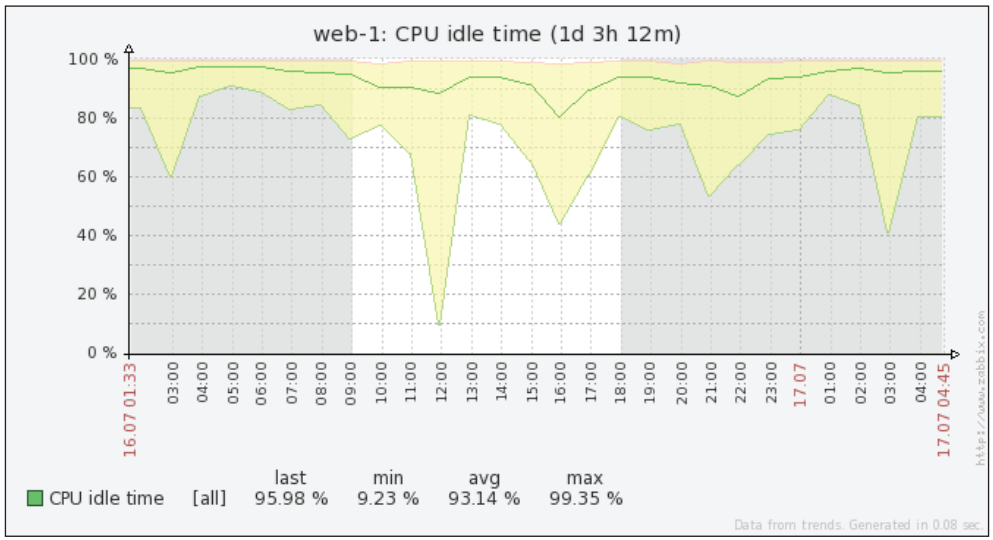


Рис. 5.3 ❖ Кратковременный пик нагрузки почти не повлиял на среднее значение

Простые графики предназначены для простого отображения и наблюдения за показателями по отдельности. Но, как вы понимаете, важно иметь возможность интерпретировать данные; например, для центрального процессора поддерживается несколько параметров, и часто весьма желательно наблюдать их все сразу на одном графике.

Ситуационные графики

В версии Zabbix 2.4 появилась совершенно новая и замечательная возможность создавать ситуационные графики, что называется, «на лету».

С ее помощью можно отобразить на одном графике сразу несколько параметров, привязанных к единой временной шкале.

Благодаря этой новой возможности любой желающий, не имеющий привилегий администратора, сможет создать требуемый график «на лету» всего несколькими щелчками мышью.

Чтобы создать ситуационный график, просто перейдите на вкладку **Monitoring ⇨ Latest data** (Мониторинг ⇨ Последние данные) и выберите элементы данных, которые вы желаете увидеть на графике, как показано на рис. 5.4.

☑ CPU interrupt time	2015-02-07 04:47:09	0.01 %	-	Graph
☑ CPU iowait time	2015-02-07 04:47:05	0.03 %	-	Graph
☑ CPU nice time	2015-02-07 04:47:03	0 %	-	Graph
☐ CPU softirq time	2015-02-07 04:47:08	0.11 %	-	Graph
☑ CPU steal time	2015-02-07 04:47:07	0 %	-	Graph
☑ CPU system time	2015-02-07 04:47:04	0.42 %	-	Graph
☑ CPU user time	2015-02-07 04:47:02	0.42 %	-0.1 %	Graph

Рис. 5.4 ❖ Выбор элементов для отображения на ситуационном графике

На той же странице, в самом низу, выберите тип графика в раскрывающемся меню – по умолчанию выбран «этажерочный» график (stacked graph)[1], но вы можете выбрать стандартный вид – и щелкните на кнопке **Go** (Вперед).

Результат нашего примера показан на рис. 5.5.

Обратите внимание, насколько быстро можно переключать вид графика.

Эта особенность не требует связывать график с определенным хостом. То есть с ее помощью можно получить график изменения параметров, поступающих с разных хостов; например, на одном графике можно отобразить изменение нагрузки на процессор сервера баз данных и сервера приложений.

Теперь можно немного углубиться в ситуационные графики и познакомиться с некоторыми интересными особенностями.

---

[1] В электронной документации Zabbix этот вид графиков называется «стекируемый график» (https://www.zabbix.com/documentation/2.4/ru/manual/config/visualisation/graphs/adhoc). – *Прим. перев.*

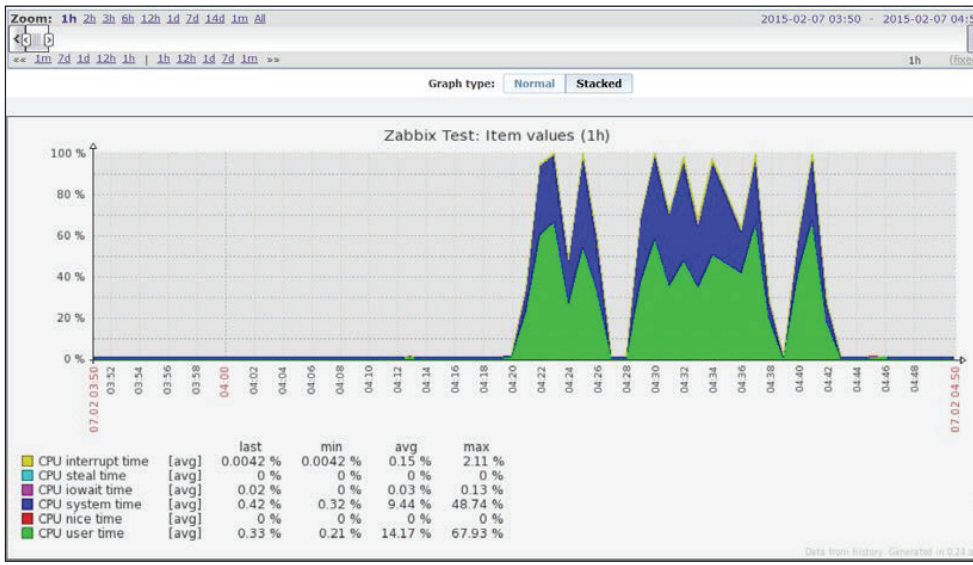


Рис. 5.5 ❖ Пример «этажерочного» графика

Особенности ситуационных графиков

Давайте теперь рассмотрим некоторые особенности, которые пригодятся вам при создании комплексных экранов.

Для ситуационных графиков Zabbix генерирует адреса URL, такие как: `http://<ZABBIX-GUI>/zabbix/history.php?sid=<SID>&form_refresh=2&action=batchgraph&itemids[23701]=23701&itemids[23709]=23709&itemids[23705]=23705&itemids[23707]=23707&itemids[23704]=23704&itemids[23702]=23702&graphtype=1&period=3600`.

Этот адрес URL включает следующие параметры:

- `sid`: представляет идентификатор сеанса и не является строго обязательным;
- `form_refresh`: определяет частоту обновления и не является строго обязательным;
- `itemids[id]=value`: представляет фактический элемент, отображаемый на графике;
- `action=[batchgraph|showgraph]`: определяет вид графика.

Обратите внимание, что вид графика легко можно переключить, заменив значение `batchgraph` (по умолчанию) в параметре `action` значением `showgraph`. Главное отличие здесь состоит в том, что на графике вида `batchgraph` отображаются только средние значения, а на графике вида `showgraph`, включающем триггеры, дополнительно отображаются максимальное и минимальное значения элементов.

Пример того же графика со значением `showgraph` в параметре `action` показан на рис. 5.6.

Здесь ясно видно, что в график включен триггер. Такой подход может очень пригодиться, особенно если вы разработчик приложений и ищете стандартные графики, не входящие в состав стандартного шаблона.

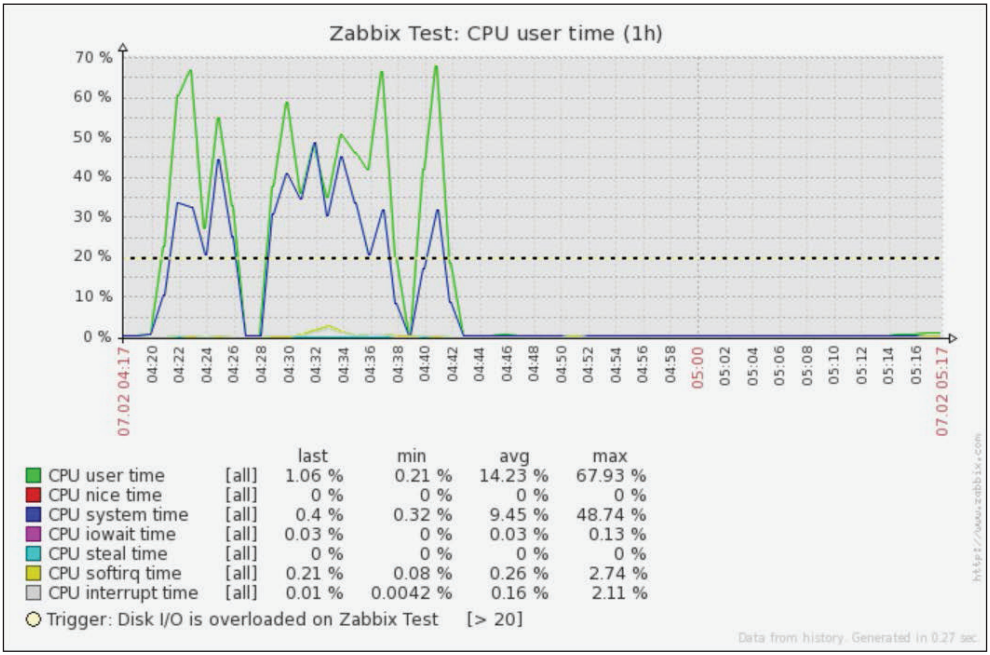


Рис. 5.6 ❖ Результат замены значения `batchgraph` значением `showgraph` в параметре `action`

А теперь рассмотрим еще одну скрытую особенность. Если потребуется получить график для повторного использования где-то в другом месте, вы можете просто использовать тот же адрес URL с теми же параметрами, но вместо страницы `history.php` указать `chart.php`. Результат такой замены показан на рис. 5.7.

Эта веб-страница отображает только сам график. Используя этот прием, вы можете создать закладки на нужные графики и вызывать их одним щелчком!

Нестандартные графики

К настоящему моменту мы познакомились только с компонентами графиков, но не обсуждали их функциональные возможности, помогающие просматривать тренды или получать детализированное отображение определенного периода времени. Zabbix предлагает поддержку нестандартных, или пользовательских, графиков, которые должны создаваться и настраиваться вручную. Например, в стандартном шаблоне **Template OS Linux** существует несколько таких предопределенных графиков. Чтобы создать нестандартный (пользовательский) график, нужно перейти на вкладку **Configuration** ⇨ **Hosts** (или **Templates**) (Настройка ⇨ Узлы сети (или Шаблоны)), щелкнуть на ссылке **Graphs** (Графики) и затем на ссылке **Create graph** (Создать график).

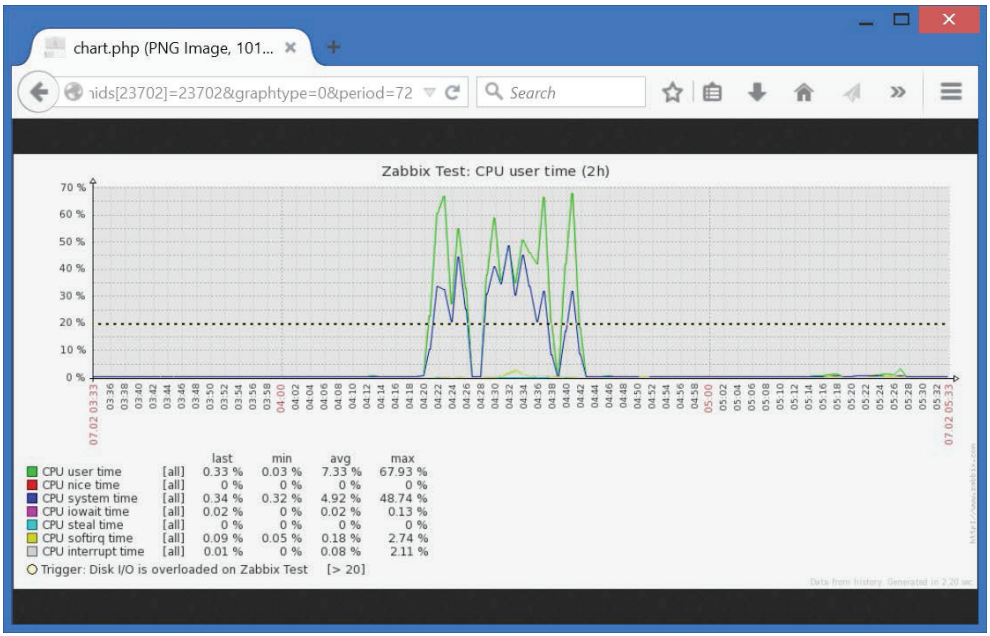


Рис. 5.7 ❖ Результат замены страницы `history.php` на `chart.php`

Обычно графики создаются в шаблонах, чтобы их проще было применить к группам серверов. Примером может служить график использования процессора **CPU utilization** в шаблоне **Template OS Linux**. Это довольно универсальный график, включающий несколько измеряемых параметров, он прекрасно подходит для применения ко всем серверам на Linux.

Графики являются мощной особенностью инфраструктуры мониторинга Zabbix. В нестандартном графике можно настроить отображение рабочего времени и легенды, используя для этого разные виды графиков. Для примера на рис. 5.8 показаны настройки графика **CPU utilization**.

Как видите, это «этажерочный» график с легендой для оси x и фиксированной шкалой по оси y. В данном конкретном случае нет смысла использовать переменный масштаб по оси y для отображения минимальных и максимальных значений, поскольку значения всех измеряемых параметров представлены в процентах, сумма которых всегда составляет 100%, как показано на рис. 5.9.

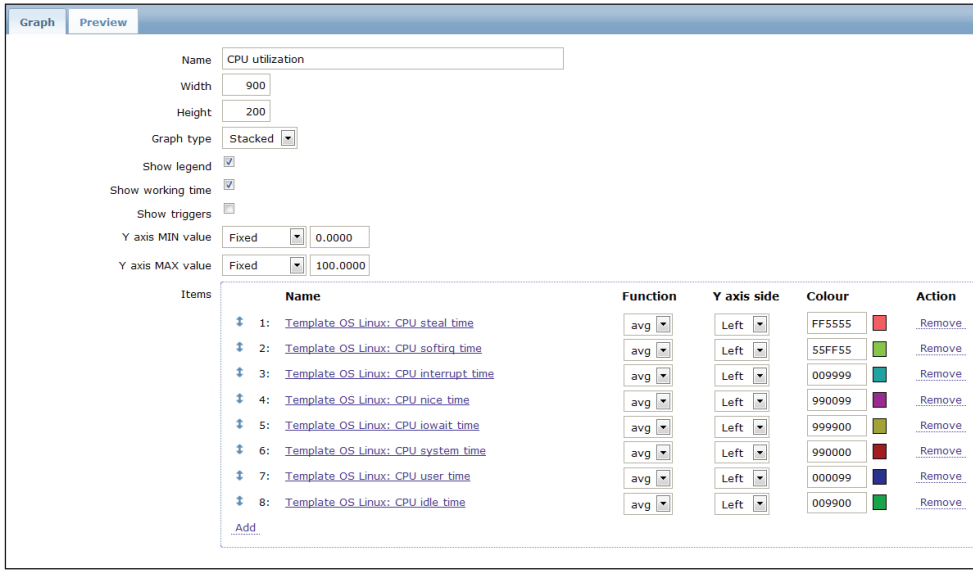


Рис. 5.8 ❖ Настройки графика **CPU utilization**

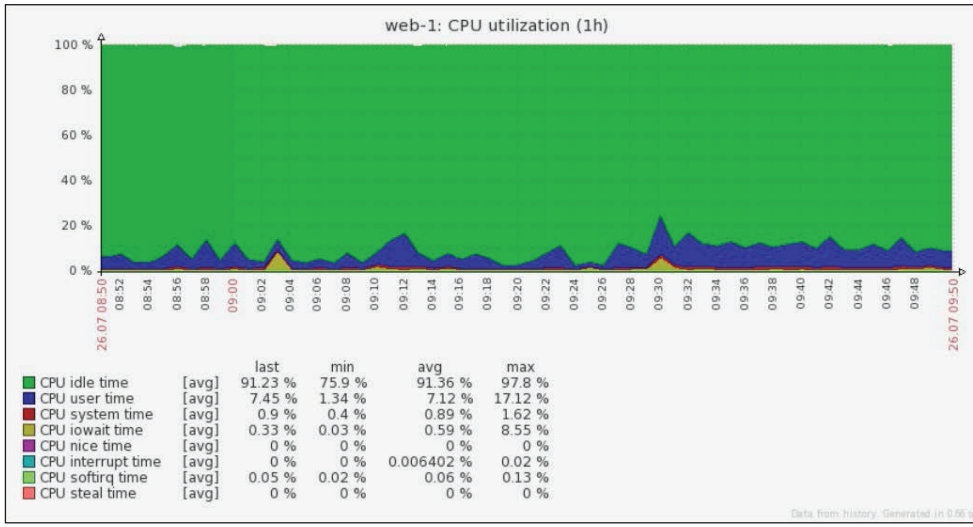


Рис. 5.9 ❖ Вид графика **CPU utilization**

В отношении триггеров и рабочего времени требуется сделать несколько замечаний. Это всего лишь два флажка в настройках, но они изменяют внешний вид графика. На предыдущем графике настроено отображение рабочих часов, но не триггеров, из-за их отсутствия для отображаемых параметров. Рабочее время, как уже отмечалось выше, выделяется белым фоном. Такое выделение действительно полезно во всех случаях, когда сервер имеет два разных режима работы или решает две разные задачи. В качестве практического примера представьте сервер, находящийся в Нью-Йорке, который контролирует все сделки на рынке США. Если рабочие часы – как в данном случае – совпадают со временем активных торгов, сервер почти наверняка все это время будет получать обновляющиеся данные. Теперь представьте, что та же торговая компания работает на азиатском рынке; ее сотрудники определенно заходят запросить сервер в Нью-Йорке, чтобы узнать, что происходило в течение рабочего дня. В таком случае сервер будет работать в двух режимах, и выделение рабочего времени на графике будет очень полезно.

Выделение рабочего времени на графике полезно еще и для того, чтобы оценить, запускались ли триггеры в это время.

Чтобы отобразить на графике триггеры, достаточно просто установить флажок **Show triggers** (Отображать триггеры), и все триггеры, которые были определены, появятся на графике. Но что делать, если линия триггера не видна на графике? Например, взгляните на рис. 5.10.

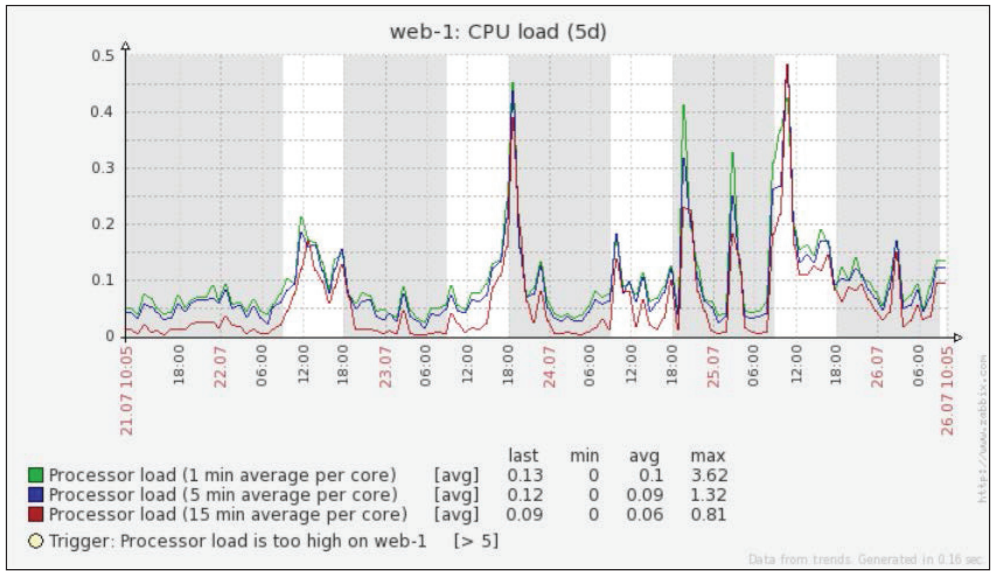


Рис. 5.10 ❖ Линия триггеров не видна на графике, оказавшись выше верхней его границы

Куда делась линия, соответствующая триггеру? Все просто. Поскольку триггер определен как уровень нагрузки на процессор, равный 5, его линия оказалась за верхней границей графика. Чтобы изменить масштаб графика по оси *y*, для отображения линии триггера нужно внести в настройки несколько изменений, в частности изменить поля **Y axis MIN value** (МИН значение оси Y) и **Y axis MAX value** (МАКС значение оси Y). По умолчанию предопределенный график нагрузки на процессор использует минимальное значение, равное нулю, а максимальное значение вычисляется динамически. Обе настройки надо изменить, как показано на рис. 5.11.

Рис. 5.11 ❖ Настройки, управляющие масштабом оси Y

Теперь обновите график, и на нем должна появиться линия триггера, которая прежде была не видна, потому что процессор все время практически простаивал и линия триггеров оказалась за верхней границей графика с автоматическим масштабированием оси Y. Результат можно наблюдать на рис. 5.12.

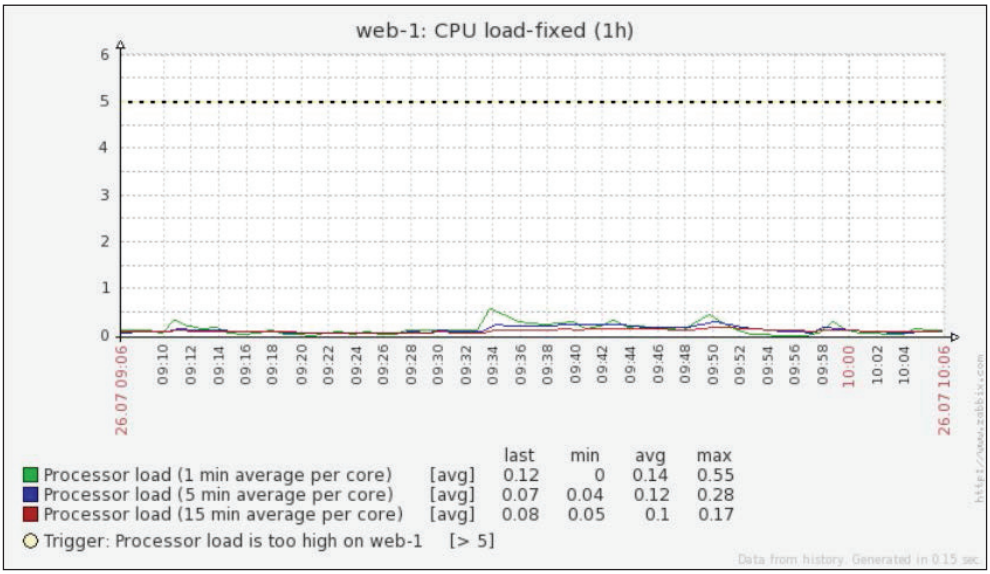


Рис. 5.12 ❖ Результат изменения масштаба оси Y

Как вы уже наверняка заметили, Zabbix не отображает периоды короче одного часа. Минимальный период, отображаемый графиком, примерно равен одному часу.

Zabbix поддерживает следующие виды пользовательских графиков:

- обычные;
- «этажерочные»;
- круговые диаграммы;
- расширенные круговые диаграммы.

Кроме того, Zabbix поддерживает разные стили оформления. В графиках, отображающих сетевой трафик, например, можно использовать градиентную заливку областей под линиями, что поможет легче отличать входящий и исходящий трафики при отображении на одном графике. Пример такого оформления показан на рис. 5.13. В данном случае отсутствует значение общей пропускной способности сети, поэтому максимальное значение по оси *y* определяется по исходящему трафику. Но на «этажерочных» графиках области складываются, поэтому график будет использовать для масштабирования предполагаемую пропускную способность.

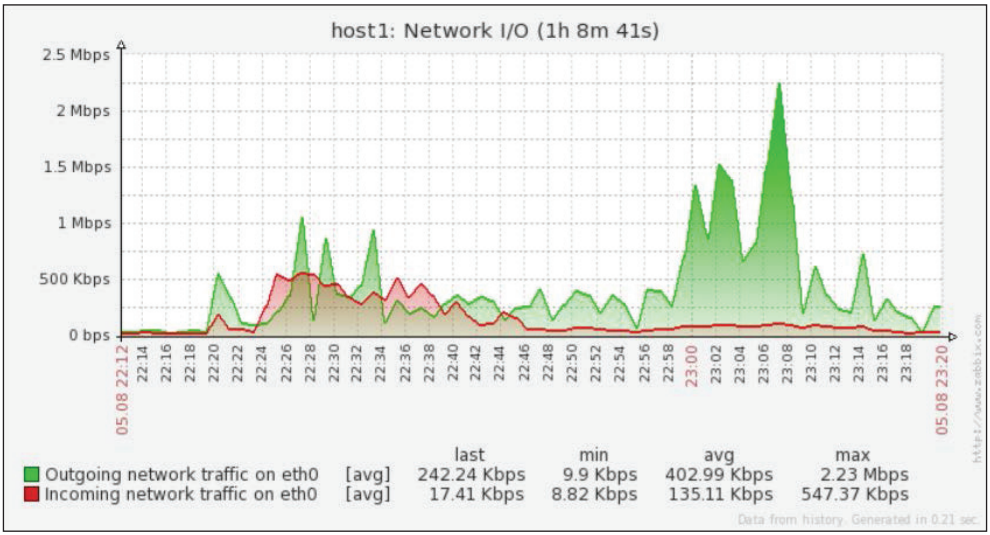


Рис. 5.13 ❖ Оформление графиков градиентной заливкой

Чтобы было понятнее, чем отличаются обычные и «этажерочные» графики, для сравнения на рис. 5.14 приводится тот же график за тот же период времени.

Как видите, пики и верхняя линия представляют суммарный трафик (входящий и исходящий), протекающий через сетевую карту. График на рис. 5.14 представляет полный сетевой трафик, обрабатываемый сетевой картой.

Обзор всех параметров настройки графиков

Zabbix – очень гибкая система, и ее графики обладают большим количеством атрибутов, позволяющих настраивать их в очень широких пределах. Все поддерживаемые атрибуты перечислены в табл. 5.1.

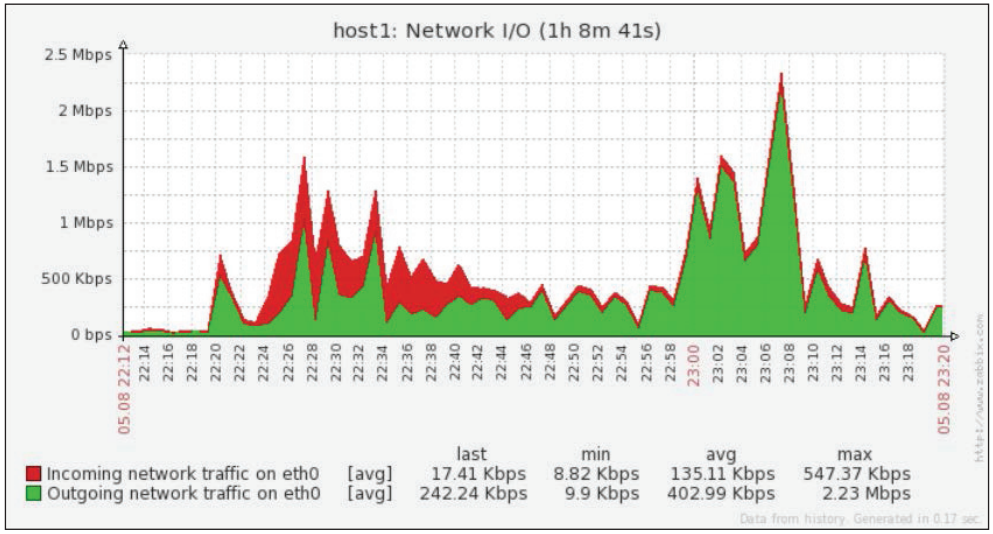


Рис. 5.14 ❖ Этажерочный график для сравнения

Таблица 5.1 ❖ Атрибуты настройки графиков

Атрибут	Описание
Name (Имя)	Имя графика (должно быть уникальным)
Width (Ширина)	Ширина графика в пикселях
Height (Высота)	Высота графика в пикселях
Graph type (Тип графика)	• **Normal** (Обычный): значения отображаются линиями, закрашенными областями, жирными линиями, точками, пунктирными линиями или градиентной заливкой; • **Stacked** (этажерочный (стекируемый)): значения отображаются как области, накладываемые сверху друг на друга; • **Pie** (круговой): значения отображаются в виде круговой диаграммы; • **Exploded** (расширенный круговой): значения отображаются как сегменты, «вырезанные» из круга
Show legend (Показывать легенду)	Если установить этот флажок, на графике будет отображаться легенда
Show working time (Отображать рабочее время)	Если установить этот флажок, рабочее время будет выделено белым фоном, а нерабочее – серым
Show triggers (Отображать триггеры)	Если установить этот флажок, на графике будет отображаться линия триггера (недоступно для кругового и расширенного кругового графиков)
Percentile line (left/right) (Процентная линия (слева/справа))	Доступно только для обычных графиков. Если установить этот флажок, на графике будет отображаться процентная линия. Если, например, задать 90%, на графике будет отображена линия на уровне 90% значений

***Таблица 5.1** (окончание)*

Атрибут	Описание
Y axis MIN/MAX value (МИН/МАКС значение оси Y)	Определяет минимальное/максимальное значение оси Y: • **Calculated** (Вычисляемое): минимальное/максимальное значение вычисляется автоматически; • **Fixed** (Фиксированное): используется фиксированное минимальное/максимальное значение; • **Item** (Элемент данных): минимальное/максимальное значение определяется последним значением выбранного элемента данных
3D View (3D-вид)	Отображает график в 3-мерном виде (только для круговых и расширенных круговых диаграмм)

В табл. 5.2 перечислены атрибуты настройки отображения элементов данных.

Таблица 5.2 ❖ Атрибуты настройки отображения элементов данных

Атрибут	Описание
Sort order (Порядок сортировки)	Определяет приоритет элемента, учитываемый при выводе. Полезно использовать для отрисовки линий или областей друг перед другом. Здесь можно перетаскивать элементы мышью, чтобы определить правильный порядок отрисовки. Элемент с приоритетом 0 обрабатывается в первую очередь. Всего поддерживается до 100 приоритетов
Name (Имя)	Имя отображаемого элемента. Имя определяется в форме `<источник>:<имя_измеряемого_параметра>`. То есть внутри конфигурации хоста имена должны иметь вид: `<имя_хоста>:<имя_измеряемого_параметра>`. Внутри шаблона имена должны иметь вид: `<имя_шаблона>:<имя_измеряемого_параметра>`
Type (Тип)	Тип, определяется только для круговых и расширенных круговых диаграмм: • **Simple** (Простой); • **Graph sum** (Суммарный график)
Function (Функция)	Определяет, какие значения будут отображаться, если элемент данных имеет несколько значений: • **All** (Все): все (минимальное, среднее и максимальное); • **Min** (Мин): только минимальное; • **Avg** (Сред): только среднее; • **Max** (макс): только максимальное
Draw style (Стиль отрисовки)	Стиль отрисовки, доступен только для обычных графиков: • **Line** (Линия): линии; • **Filled region** (Заполнение): области с заливкой; • **Bold line** (Жирная линия): толстые линии; • **Dot** (Точечная линия): точки; • **Dashed line** (Пунктирная линия): пунктирные линии
Y axis side (Расположение оси Y)	Доступно только для «этажерочных» и обычных графиков и определяет, с какой стороны будет нарисована ось *y*
Colour (Цвет)	Помимо стандартных цветов, доступных в палитре, можно определить свой цвет, указав его в шестнадцатеричном формате RGB

Вы можете смело экспериментировать с этими атрибутами. В версии Zabbix 2.0 имеется вкладка **Preview** (Предварительный просмотр), которую очень удобно использовать при настройке графиков внутри конфигурации хоста. Если вы конструируете график в конфигурации шаблона, эта вкладка будет

бесполезна из-за отсутствия данных для отображения. При работе над шаблоном лучше всего использовать два окна, чтобы видеть производимые изменения, обновляя страницу (клавишей **F5**) для хоста, который наследует настраиваемый шаблон.

Все параметры, описанные выше, как вы уже наверняка осознали, действительно обеспечивают большую гибкость в создании графиков.

На графике можно отобразить не более трех линий триггеров, а если график имеет размер меньше 120 пикселей, триггеры вообще не отображаются; поэтому внимательно отнеситесь к настройке размеров своих графиков и обязательно проверяйте все вносимые изменения.

Визуализация данных с применением карт

В Zabbix имеет мощный инструмент отображения данных с географической привязкой, помогающий создавать самые настоящие карты. Карта – это графическое представление инфраструктуры, в котором можно изобразить сервер, компоненты сети и взаимосвязи между ними.

Отличительной чертой поддержки карт в Zabbix является полная динамичность, благодаря которой можно отображать на карте активные предупреждения, извещения и триггеры с использованием разных пиктограмм, цветов и надписей. Это позволяет создать очень информативное представление вычислительного центра или службы. На карту можно поместить следующие элементы:

- хосты;
- группы хостов;
- триггеры;
- изображения;
- карты.

Все эти элементы динамически изменяются триггерами или макросами и таким способом представляют информацию о состоянии карт и их элементов.

Чтобы пользователь мог создавать или настраивать карты, он должен обладать привилегиями администратора. То есть в Zabbix отсутствует отдельная роль, предназначенная для создания карт. Кроме того, пользователь должен иметь право на чтение/запись для всех хостов, которые должны быть представлены на карте. Это означает, что нет никакого способа ограничить привилегии разработчика карт, но вы можете ограничить круг хостов, доступных такому пользователю, объединив их в группу.

На рис. 5.15 показан пример карты, которую легко воспроизвести в Zabbix.

На рис. 5.16 можно видеть комбинацию пиктограмм в кружочках с дополнительной информацией в подписях. Чтобы иметь полное представление о том, что отображает эта карта, необходимо знать, как Zabbix интерпретирует хосты и триггеры на карте. На рис. 5.16 приводятся возможные комбинации триггеров с разными уровнями важности.

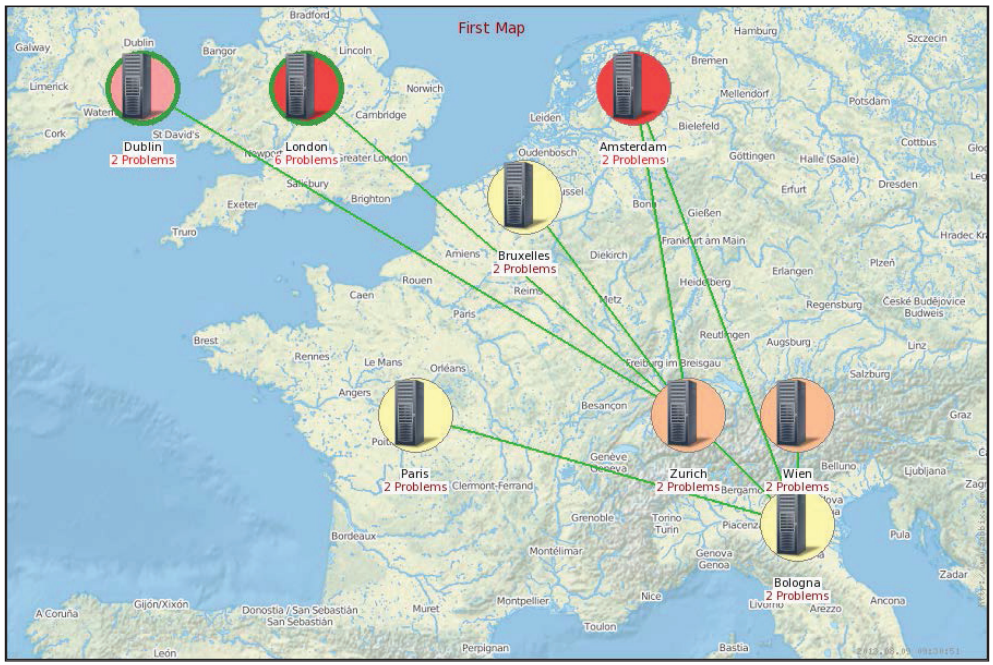


Рис. 5.15 ❖ Пример карты

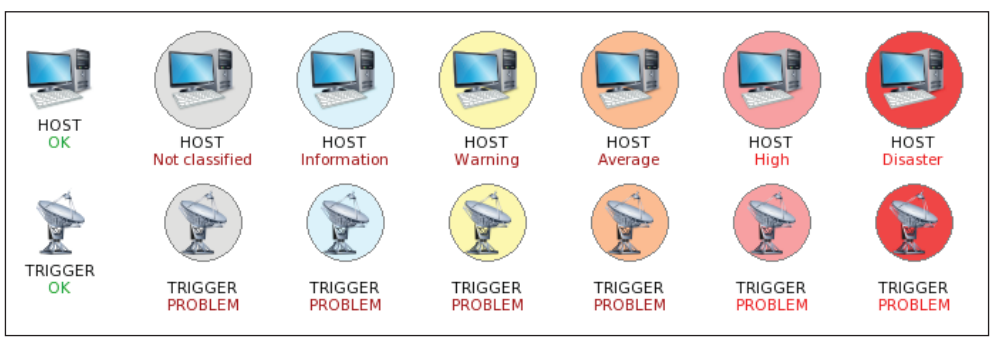


Рис. 5.16 ❖ Разные комбинации состояний триггеров

На рис. 5.16 показаны, слева направо:

- хост, не имеющий триггера;
- хост с триггером, уровень важности которого определяется как «не классифицировано»;
- хост с триггером, уровень важности которого определяется как «информация»;
- хост с триггером, уровень важности которого определяется как «предупреждение»;

- хост с триггером, уровень важности которого определяется как «средний»;
- хост с триггером, уровень важности которого определяется как «высокий»;
- хост с триггером, уровень важности которого определяется как «чрезвычайный».

Пиктограммы в нижнем ряду, соответствующие триггерам, имеют ту же классификацию. Уровень важности выражается цветом фона, а не подписью под меткой «HOST». Подписи красного цвета на рис. 5.16 – это имена триггеров. Имена триггеров на рис. 5.16 выбраны исключительно для наглядности, чтобы показать их классификацию. Обратите внимание, что под подписью «TRIGGER» на рис. 5.16 отображается состояние триггера, а не его имя, как в случае с хостами.

Если триггер изменится, его состояние будет отображено, как показано на рис. 5.17.

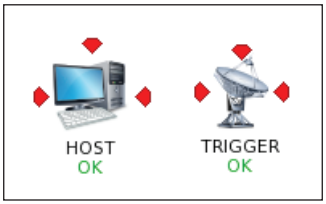


Рис. 5.17 ❖ Состояние триггера

Если на хосте возникнет проблема и сработает триггер, он будет отображен, как показано на рис. 5.18.

Рис. 5.18 ❖ Состояние хоста

Обратите внимание, что в данном случае пиктограмма отображается со стрелками, направленными на нее. Эти стрелки сообщают, что триггер только что изменил состояние. На рис. 5.19 показан хост, на котором обнаружено шесть проблем.

Как видите, в данном случае имеется несколько триггеров, обнаруживших проблемы. Триггер с наибольшей важностью определяет цвет кружка, в который заключена пиктограмма. После квитирования всех триггеров вокруг кружка появится зеленое кольцо, как показано на рис. 5.20.

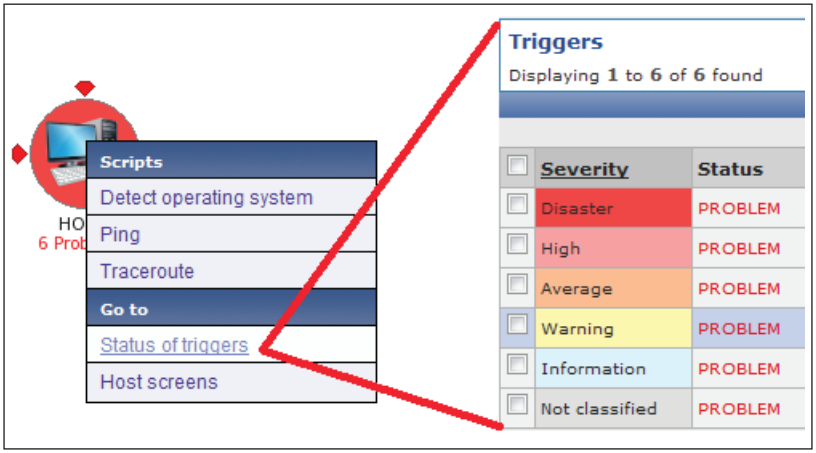


Рис. 5.19 ❖ Проблемы, обнаруженные на хосте

Рис. 5.20 ❖ Отображение пиктограммы после квитирования триггеров

Рядом со второй пиктограммой (рис. 5.20) выводится дополнительная информация о проблемах и количество неквитированных триггеров, благодаря чему всегда видно, сколько проблем обнаружено и сколько из них еще не квитировано.

Третья пиктограмма с квадратным фоном – это хост, определение состояния которого выключено, о чем сообщает серый цвет; как только этот хост станет недостижимым в сети, фон окрасится в красный цвет.

Создание первой карты в Zabbix

Чтобы создать новую карту, перейдите на вкладку **Configuration** ⇨ **Maps** ⇨ **Create map** (Настройка ⇨ Карты сети ⇨ Создать). В результате откроется форма, изображенная на рис. 5.21.

Названия большинства атрибутов говорят сами за себя; в поле **Name** (Имя) должно быть определено уникальное имя карты, а значения полей **Width** (Ширина) и **Height** (Высота) выражаются в пикселях.

Если сначала определить карту слишком большого размера, а потом попытаться уменьшить ее, некоторые хосты могут оказаться за границами и станут не-

видимы. Если это случится, не волнуйтесь, ничего не потеряно. Они все еще находятся на карте, просто невидимы. Они появятся вновь после восстановления прежних размеров карты.

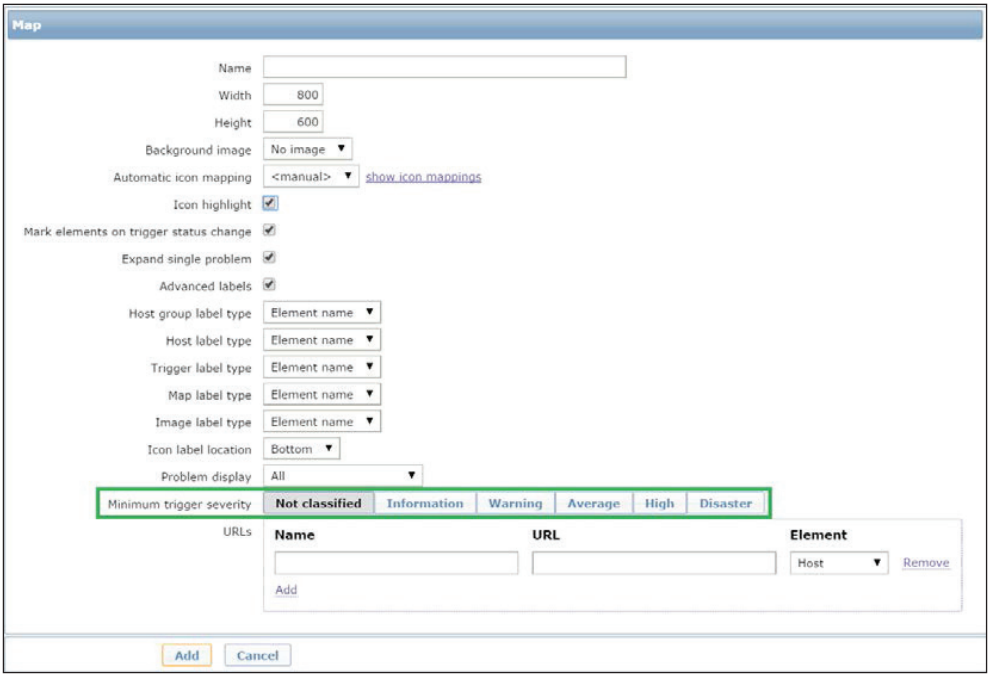


Рис. 5.21 ❖ Форма создания новой карты

Теперь рассмотрим другие параметры.

- **Background image** (Фоновое изображение): здесь можно определить фоновое изображение для карты.

В Zabbix отсутствуют фоновые изображения по умолчанию. Чтобы добавить свое изображение, перейдите на вкладку **Administration** ⇨ **General** (Администрирование ⇨ Общие), выберите пункт **Images** (Изображения) в списке и не забудьте в поле **Type** (Тип) выбрать **Background** (Фоновое изображение), а не **Icon** (Иконка). Отличные бесплатные карты можно найти на сайте www.openstreetmap.org.

- **Automatic icon mapping** (Автоматическое соответствие иконок): этот флажок позволяет отображать некоторые пиктограммы в соответствии с полями инвентарных данных узлов сети. Их можно определить на вкладке **Administration** ⇨ **General** ⇨ **Icon mapping** (Администрирование ⇨ Общие ⇨ Соответствие иконок).
- **Icon highlight** (Подсветка иконок): если вы установите этот флажок, пиктограммы получат фоновый кружок, цвет которого будет соответствовать триггеру с наивысшей важностью.

- **Mark elements on trigger status change** (Помечать элементы при изменении состояния триггера): этот флажок отвечает за визуализацию недавнего изменения состояния триггера (красными стрелками, как на рис. 5.18).

Стрелки отображаются только в течение 30 минут, после чего они исчезают, а измененное состояние триггера становится новым нормальным состоянием.

- **Advanced labels** (Расширенные подписи): этот флажок дает возможность указать типы подписей для всех элементов на карте. То есть для каждого вида элементов – хоста, группы хостов, триггера, карты и изображения – можно определить свой тип подписи. Поддерживаются следующие типы подписей:
  - **Label** (Подпись): подпись к пиктограмме;
  - **IP Address** (IP-адрес): IP-адрес хоста (доступен только в конфигурациях хостов);
  - **Element name** (Имя элемента): имя элемента, например имя хоста;
  - **Status only** (Только состояние): отображает только состояние, то есть надпись OK (OK) или PROBLEM (ПРОБЛЕМА);
  - **Nothing** (Ничего): означает отсутствие подписи;
  - **Custom label** (Пользовательская подпись): определяется пользователем (допускается использовать макросы).
- **Icon label location** (Расположение подписи к иконке): это поле определяет местоположение всех подписей по умолчанию. Предлагается выбор из следующих вариантов: **Bottom** (По нижнему краю), **Left** (По левой стороне), **Right** (По правой стороне) и **Top** (По верхнему краю).
- **Problem display** (Отображение проблем): список с вариантами:
  - **All** (Все): отображать полное количество проблем;
  - **Separated** (Неподтвержденные отдельно): отображать количество неподтвержденных (неквитированных) проблем отдельно от общего числа проблем;
  - **Unacknowledged** only (Только неподтвержденные): отображать только количество неподтвержденных (неквитированных) проблем.
- **URLs** (URL'ы): здесь можно указать адреса URL для всех видов элементов (с подписями). Они будут отображаться как ссылки и допускают применение макросов, например `{MAP.ID}`, `{HOSTGROUP.ID}`, `{HOST.ID}` и `{TRIGGER.ID}`.

В версии Zabbix 2.2 появилась новая возможность, позволяющая определять наименьшую важность триггера.

После определения этого параметра настройки на карте будут отображаться только триггеры с указанной или более высокой важностью; это помогает уменьшить количество отображаемых триггеров, так как все триггеры с более низкой важностью отображаться не будут. Область настройки этого параметра выделена зеленой рамкой на рис. 5.21.

Уровень важности, выбранный в конфигурации карты, можно переопределить при просмотре на вкладке **Monitoring** ⇨ **Maps** (Мониторинг ⇨ Карты сетей),

выбрав желаемую важность в параметре **Minimum severity** (Минимальная важность), как показано на рис. 5.22.

Рис. 5.22 ❖ Минимальный уровень серьезности можно изменить при просмотре

Важные замечания о макросах и адресах URL

Параметр настройки **URLs** (URL'ы) дает дополнительные интересные возможности, но его объяснение желательно проводить на примере, потому что иначе его трудно понять.

Итак, увидев сработавший триггер с высокой важностью, типичное первое действие – проверить последние данные мониторинга, получаемые с хоста, или отобразить комплексный экран, объединяющий графики, триггеры и данные, которые помогут проанализировать случившееся на первом уровне. Поэтому, с практической точки зрения, было бы полезно предоставить ссылку, ведущую к комплексному экрану с последними данными, полученными с хоста, чтобы ею можно было воспользоваться после выделения сервера на карте как проблемного. Чтобы автоматизировать эту задачу и уменьшить число щелчков мышью, можно просто скопировать ссылку на требуемую страницу; например, ссылка на последние данные могла бы иметь вид: `http://<ZABBIX-SERVER>/zabbix/latest.php?sid=eec82e6bdf51145f&form_refresh=1&groupid=2&hostid=10095`.

Чтобы автоматизировать переход к последним данным, нужно заменить переменные части этого адреса URL макросами.

Параметр запроса `sid` в URL представляет идентификатор сеанса (session ID); он передается с целью защиты от атак вида «подделка межсайтовых запросов». Этот параметр запроса можно опустить. Параметр `groupid` в данном примере также можно опустить. В результате URL сокращается до `http://<ZABBIX-SERVER>/zabbix/latest.php?form_refresh=1&hostid=10095`.

Теперь можно без труда определить ссылку – достаточно лишь заменить значение параметра `hostid` макросом `{HOST.ID}`: `http://<ZABBIX-SERVER>/zabbix/latest.php?form_refresh=1&hostid={HOST.ID}`.

И настроить URL, как показано на рис. 5.23.

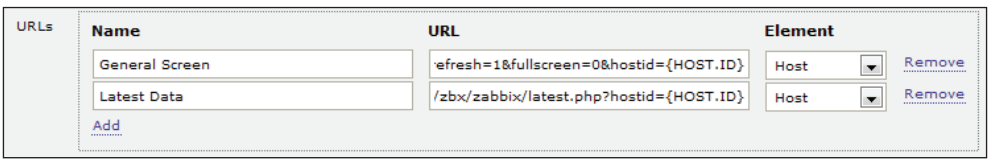


Рис. 5.23 ❖ Настройка адреса URL для ссылки на страницу с последними данными

На рис. 5.23 показана ссылка на общий комплексный экран **General Screen**, где собраны наиболее важные графики. Адрес URL `http://<ZABBIX-SERVER>/zabbix/screens.php?sid=eec82e6bdf51145f&form_refresh=1&fullscreen=0&elementid=17&groupid=2&hostid=10094&period=3600&stime=20140807161304` – это адрес страницы с экраном для данного хоста.

В предыдущем URL, как и прежде, можно опустить параметр `sid` и параметр `period`. В отсутствие последнего на экране будут отображаться данные за последний час. Точно так же можно опустить параметры `stime` и `groupid`. В результате сокращенный адрес URL получит вид: `http://<ZABBIX-SERVER>/zabbix/screens.php?form_refresh=1&fullscreen=0&hostid=10094&groupid=2`.

Чтобы сделать его динамическим, значения параметров `hostid` и `groupid` следует заменить макросами: `http://<ZABBIX-SERVER>/zabbix/screens.php?form_refresh=1&fullscreen=0&hostid={HOST.ID}&groupid={HOSTGROUP.ID}`.

Результат этой настройки показан на рис. 5.24.

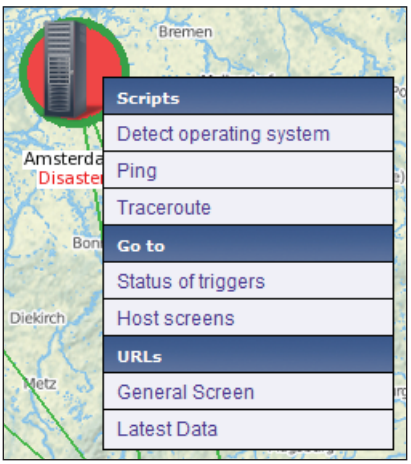


Рис. 5.24 ❖ Результат настройки адреса URL

Как видите, щелкнув на пиктограмме хоста, где обнаружились проблемы, вы получаете две новые ссылки – **Latest Data** и **General Screen**, – которые динамически создаются для всех хостов.

Это позволяет создавать зависимости вида «главный/подчиненный» (master/detail). В данном случае главной является карта, а подчиненным может быть, например, экран или окно с последними данными. Таким способом можно создавать собственные меню, запускающие сценарии или открывающие страницы с более подробной информацией, которая может использоваться для анализа проблем.

Чтобы добавить сценарий для запуска на выбранном хосте, перейдите на вкладку **Administration** ⇨ **Scripts** (Администрирование ⇨ Скрипты).

Внутри карты

Закончив настройки, описанные выше, можно приступать к самой приятной части конфигурирования карты – добавлению элементов. Инструменты управления, доступные внутри карты, очень просты и понятны, как показано на рис. 5.25.

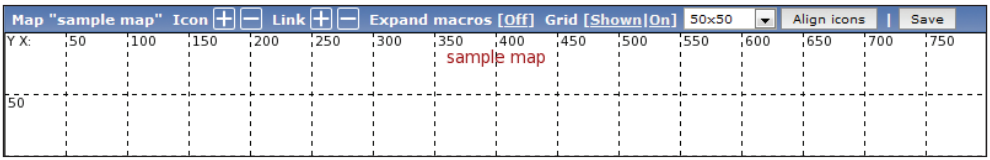


Рис. 5.25 ❖ Внутри карты

Чтобы добавить новый элемент, щелкните на кнопке со знаком «плюс» (+), а чтобы удалить выбранный – на кнопке со знаком «минус» (–). Новые элементы появляются в верхнем левом углу карты. Если щелкнуть на пиктограмме, появится конфигурационная панель, как показано на рис. 5.26.

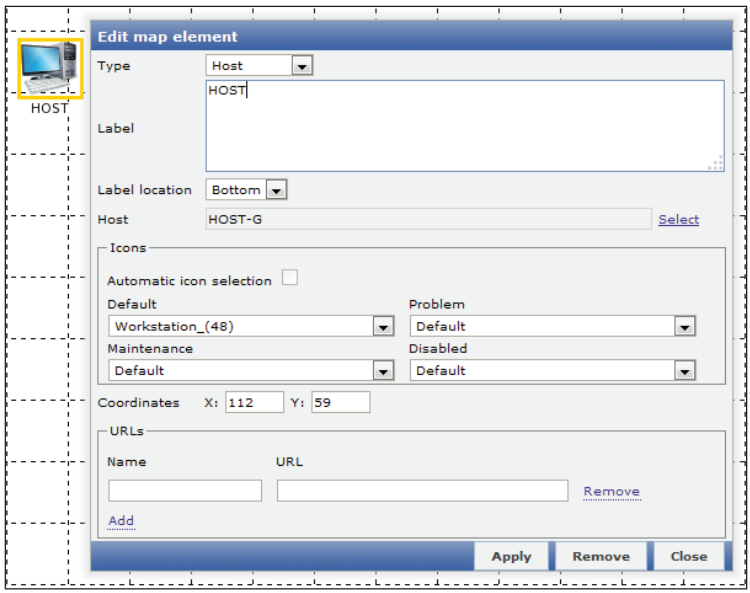


Рис. 5.26 ❖ После щелчка на пиктограмме появляется конфигурационная панель

По умолчанию создается элемент типа **Icon** (Иконка). На рис. 5.26 после создания элемента был выбран тип **Host** (Узел сети), но вообще на выбор доступны следующие типы:

- **Host** (Узел сети): эта пиктограмма отображает состояние всех триггеров выбранного узла сети;

- **Map** (Карта сети): эта пиктограмма отображает состояние всех элементов на карте;
- **Trigger** (Триггер): эта пиктограмма отображает состояние одного выбранного триггера;
- **Host group** (Группа узлов сети): эта пиктограмма отображает состояние всех триггеров всех хостов в выбранной группе;
- **Image** (Изображение): простое изображение, не связанное ни с каким источником данных (триггером, хостом или чем-то еще).

Поле ввода **Label** (Подпись) – еще один важный параметр настройки элемента. Здесь можно ввести произвольный текст и использовать макросы.

Следующее поле отличается для элементов разных типов и может быть одним из следующих:

- **Host** (Узел сети): позволяет выбрать конкретный хост;
- **Map** (Карта сети): позволяет выбрать конкретную карту;
- **Trigger** (Триггер): позволяет выбрать конкретный триггер;
- **Host group** (Группа узлов сети): позволяет выбрать конкретную группу хостов;
- **Icon** (Иконка): (по умолчанию) позволяет выбрать пиктограмму для отображения.

В поле **Host group** (Группа узлов сети) можно сгруппировать все хосты по местоположению, например по городам, странам или континентам. Это позволит сгруппировать все триггеры по географическому признаку. Можно также добавить собственный URL.

Хосты и триггеры уже рассматривались выше и не требуют дополнительных пояснений. Труднее всего, пожалуй, понять, зачем вставлять в карту другие карты. Этот прием позволяет эффективно создавать обобщенные карты с пиктограммами, представляющими более детальные карты отдельных регионов или стран и позволяющими переходить на все более детальные уровни представления до окончательного пункта назначения; например, вообразите возможность такого перехода из карты страны к карте города, затем к вычислительному центру и, наконец, к конкретной стойке с сервером.

Элемент **Icon** (Иконка) внутри карты – это изображение, которому можно присвоить адрес URL. Его назначение – дать возможность добавить в карту графический элемент с адресом URL, позволяющий переходить к требуемой карте.

Ниже этого поля находится раздел **Icons** (Иконки). Здесь можно отметить флажок **Automatic icon selection** (Автоматический выбор иконки). В этом случае выбор отображаемой пиктограммы будет управляться функцией соответствия иконок.

Определив соответствие иконок в конфигурации карты, вы избавите себя от лишних щелчков мышью. Так, например, можно определить стандартные пиктограммы для хостов, и они будут использоваться автоматически.

Если вы еще не определили соответствие иконок или желаете, чтобы элемент на карте имел отличительный внешний вид, определите в следующих полях пиктограммы, которые должны использоваться для его представления:

- **Default** (По умолчанию);
- **Problem** (Проблема);
- **Maintenance** (Обслуживание);
- **Disable** (Деактивировано).

В разделе **Coordinates** (Координаты) настраивается точное местоположение элемента на карте в пикселях. И в поле **URLs** (URL'ы) можно настроить специализированный адрес URL для данного элемента.

Представьте, что вы создали разные комплексные экраны (экраны обсуждаются ниже в этой главе): в одном собраны все графики и триггеры, связанные с мониторингом сервера приложений, а в другом – все результаты мониторинга базы данных. Если элемент на карте представляет хост с действующей СУБД, можно определить URL для перехода к экрану с информацией мониторинга СУБД, если он представляет хост с сервером приложений – к экрану с информацией мониторинга сервера приложений, и т. д.

Как видите, элементы представляют большой интерес и позволяют сделать карту очень полезным инструментом для группы поддержки.

Выбор элементов

В конфигурации карты можно выбрать сразу несколько элементов. Для этого сначала нужно выбрать первый элемент, а затем, удерживая нажатой клавишу **Ctrl** (или **Shift**), выбрать остальные. Множественный выбор можно также произвести, выделив мышью область на карте, при этом будут выбраны все элементы, попавшие в прямоугольник выделения.

После выбора нескольких элементов автоматически открывается диалог **Mass update elements** (Массовое обновление элементов), как показано на рис. 5.27.

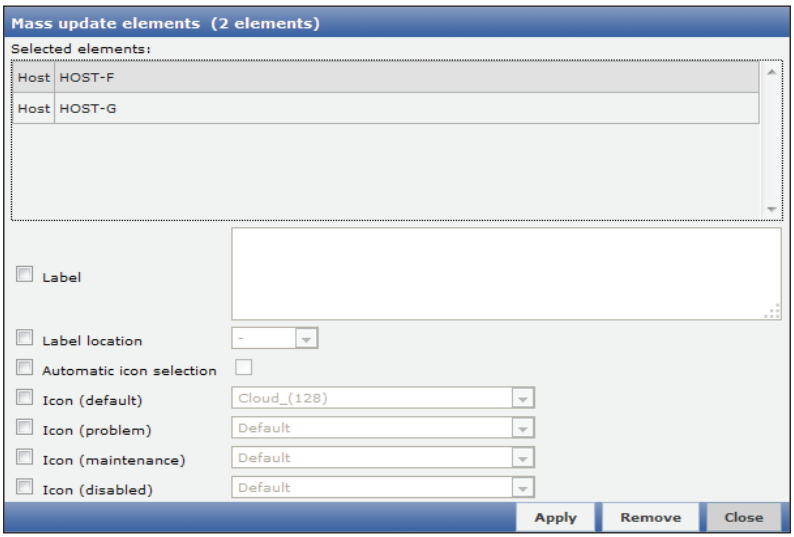


Рис. 5.27 ❖ Диалог массового обновления элементов

Здесь сразу для всех выбранных элементов можно изменить пиктограммы, подписи и местоположение подписей.

В случае изменения всех подписей настоятельно рекомендуется использовать макросы.

Теперь пришло время проложить связи между серверами на карте, чтобы они отражали физические связи. Чтобы создать связь между двумя хостами, достаточно выбрать хосты и щелкнуть на кнопке **+** рядом с подписью **Link** (Связь) в панели инструментов.

Сразу под диалогом **Mass update elements** (Массовое обновление элементов) появится диалог **Links between the selected elements** (Индикаторы связей между выбранными элементами), как показано на рис. 5.28.

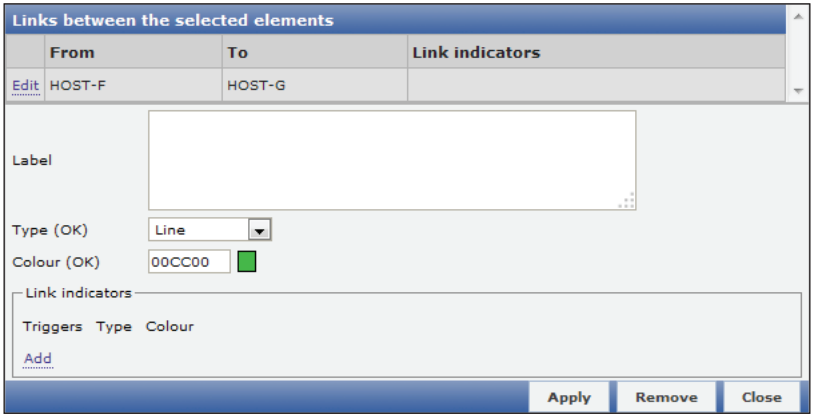


Рис. 5.28 ❖ Диалог определения индикаторов связей между выбранными элементами

Индикатор связи можно дополнить подписями, а также изменить ее внешний вид. В поле **Type (OK)** (Тип (OK)) можно выбрать тип линии: **Line** (Линия), **Bold line** (Жирная линия), **Dot** (Точечная линия) и **Dashed line** (Пунктирная линия).

Имейте в виду, что индикатор связи можно подключить к триггеру, поэтому цвет линии будет изменяться при срабатывании триггера.

Индикатор связи можно подключить к нескольким триггерам и для каждого определить свой цвет, чтобы визуально можно было сразу определить, какой триггер сработал.

Использование макросов в картах

Выше мы обсуждали параметр настройки **Label** (Подпись), с помощью которого можно настраивать подписи. Я думаю, что эксперименты с ним помогут понять, какой широтой возможностей обладает этот параметр и как можно использовать его для улучшения карт. Например, попробуйте использовать макросы в нем. У вас могут быть вполне определенные требования, такие как отображение

в подписи имени хоста, IP-адреса, количества событий триггеров (квитированных и неквитированных) и объема сетевого трафика, протекающего через сетевые интерфейсы.

Удовлетворение этих требований может показаться сложной задачей. Впрочем, так оно и есть на самом деле. Но если вы знакомы с макросами, вы без труда справитесь с ней. В этом вам поможет следующий код:

```
{HOSTNAME}
{HOST.CONN}
trigger events ack: {TRIGGER.EVENTS.ACK}
trigger events unack: {TRIGGER.EVENTS.UNACK}
Incoming traffic: {{HOSTNAME}:net.if.in[eth0].last(0)}
Outgoing traffic: {{HOSTNAME}:net.if.out[eth0].last(0)}
```

Первый макрос, `{HOSTNAME}`, отображает имя выбранного хоста. Второй макрос, `{HOST.CONN}`, отображает IP-адрес. Информацию о событиях триггеров, то есть количество квитированных и неквитированных, отображают следующие две строки с макросами: `{TRIGGERS.EVENTS.ACK}` и `{TRIGGER.EVENTS.UNACK}`. Последние две строки – самые интересные. Каждая из них включает композицию из двух вложенных макросов.

В частности, чтобы отобразить объем входящего трафика на первом сетевом интерфейсе, можно потребовать от Zabbix извлечь последнее значение из элемента `net.if.in[eth0]`. Для вычисления этого выражения требуется имя хоста, поэтому, чтобы воспользоваться этим макросом, нужно добавить к нему имя хоста, например `HOST-A`, или макрос `{HOSTNAME}`, как в данном примере.

Последний недостающий фрагмент, необходимый серверу Zabbix, чтобы получить требуемое значение, – это имя хоста. Как отмечалось выше, для этой цели можно использовать макрос `{HOSTNAME}`. То есть законченное выражение приобретает вид:

```
Incoming traffic: {{HOSTNAME}:net.if.in[eth0].last(0)}
```

Выражение для получения исходящего трафика определяется аналогично, с той лишь разницей, что используется элемент `net.if.out[eth0]`. Результат такого определения подписи показан на рис. 5.29.

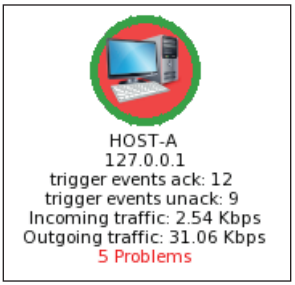


Рис. 5.29 ❖ Результат определения подписи с макросами

Используйте {HOSTNAME} или {HOST.NAME} в подписях и везде, где возможно, так как это упростит массовое изменение.

Это весьма удобный способ получить всю необходимую информацию одним взглядом на карту, без лишних щелчков мышью. В данном примере используется функция last() для получения последнего значения элемента, но точно так же можно использовать любые другие поддерживаемые функции: last(), min(), max() и avg().

Аналогичным способом можно использовать макросы в индикаторах связей, как, например, показано на рис. 5.30.

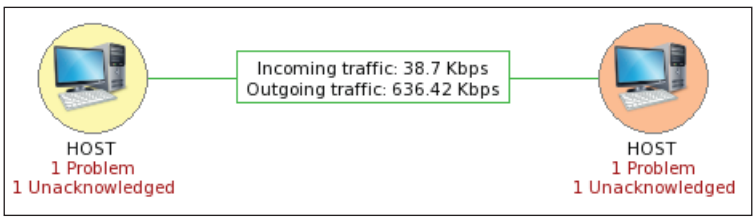


Рис. 5.30 ❖ Результат использования макросов в определении подписи для индикатора связи

Сведения о трафике в подписи на рис. 5.30 получены способом, описанным выше. Применение макросов в картах поможет вам сделать ваши карты динамическими, информативными и более привлекательными.

Комплексные экраны

В предыдущем разделе обсуждалось добавление адресов URL и ссылок на комплексные экраны. Теперь подошло время познакомиться с этими экранами поближе. Экраны просты в создании и интуитивно понятны при наблюдении за ними. В действительности комплексный экран – это страница, отображающая комплекс элементов Zabbix (отсюда и название *комплексный экран*), таких как графики, карты и текст. Одно из важнейших отличий экранов от карт состоит в том, что на карты можно поместить не все виды элементов. Например, на карту нельзя добавить график или триггер. Карты и экраны преследуют разные цели. В экран можно поместить любые элементы, характерные для данного сервера, чтобы получить обзорную картину о его состоянии.

Создание экрана

Чтобы создать экран, перейдите на вкладку **Configuration ⇨ Screen ⇨ Create** (Настройка ⇨ Комплексные экраны ⇨ Создать комплексный экран). В появившейся форме вам будет предложено указать имя экрана и его начальный размер в столбцах и строках. После этого будет выполнен переход в настройки вновь созданного экрана.

Здесь вы наверняка заметите отсутствие кнопки **Save** (Сохранить). Она не нужна просто потому, что сохранение экрана выполняется автоматически после каждой операции, например после добавления графика. Экран своей структурой напоминает таблицу (фактически он и является таблицей), как показано на рис. 5.31.

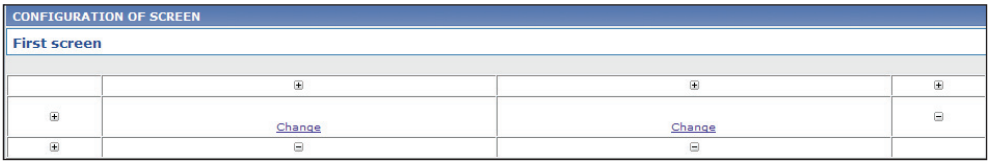


Рис. 5.31 ❖ Экран – это таблица

Если вам понадобятся дополнительные столбцы или строки, просто щелкните на кнопке + над столбцом или слева от строки, в зависимости от того, что нужно добавить – столбец или строку, соответственно.

На экраны можно добавлять элементы следующих типов:

- **Action log** (Журнал действий): отображает историю недавних событий;
- **Clock** (Часы): отображает аналоговые или цифровые часы, которые показывают текущее время на сервере или местное время;
- **Data overview** (Обзор данных): отображает последние данные для группы хостов;
- **Graph** (График): отображает один пользовательский график;
- **History of events** (История событий): отображает *n* строк (вы можете указать количество строк) с последними событиями;
- **Host group issues** (События в группах узлов сети): отображает состояния триггеров в группе хостов;
- **Host issues** (События узлов сети): отображает состояния триггеров для отдельного хоста;
- **Hosts info** (Информация об узлах сети): отображает обзорную информацию об узлах сети;
- **Map** (Карта сети): отображает единственную карту;
- **Plain text** (Простой текст): отображает простые текстовые данные;
- **Screen** (Комплексный экран): отображает другой экран (комплексный экран может содержать другие экраны);
- **Server info** (Информация о сервере): отображает обзорную информацию о сервере;
- **Simple graph** (Простой график): отображает единственный простой график;
- **Simple graph prototype** (Прототип простого графика): отображает простой график на основе элемента данных, сгенерированного правилом низкоуровневого обнаружения;
- **System status** (Состояние системы): отображает состояние системы (близко напоминает приборную панель);
- **Triggers info** (Информация о триггерах): отображает обзорную информацию о триггерах;

- **Triggers overview** (Обзор триггеров): отображает состояние триггеров для выбранной группы хостов;
- **URL**: позволяет добавить информацию из внешнего источника.

Все эти элементы имеют два общих параметра настройки – количество занимаемых строк и столбцов. Изменяя количество занимаемых столбцов, можно расширить размер ячейки так, что она будет охватывать указанное количество столбцов. То же относится к количеству занимаемых строк. Например, если в таблице с двумя столбцами указать, что элемент занимает два столбца, соответствующая ячейка будет помещена в центр таблицы по горизонтали и займет всю ее ширину. Эту возможность очень удобно использовать для создания заголовков экранов.

После добавления элемента в экран его можно переместить в любое место мышью, при этом все настройки элемента сохраняются.

Вы без опаски можете двигать элементы по экрану, их настройки при этом не теряются.

Большинство элементов являются статическими, в том смысле, что они не применяются динамически ко всем хостам и группам.

Динамические элементы

Zabbix поддерживает несколько очень удобных динамических элементов:

- графики (пользовательские);
- прототипы графиков (пользовательских);
- простые графики;
- прототипы простых графиков;
- адреса URL;
- простой текст.

Элементы прототипов графиков (пользовательских) создаются правилами низкоуровневого обнаружения (Low-Level Discovery, LLD). Элементы прототипов простых графиков также создаются правилами низкоуровневого обнаружения. В процессе мониторинга ячейка экрана отображает график на основе элемента, созданного правилом LLD. Обратите внимание, что если элемент не сгенерирован, в ячейке ничего не отображается.

Также, начиная с версии Zabbix 2.4, адреса URL превратились в динамические элементы. Для поддержки этой новой функциональной возможности в поле **URL** разрешается использовать следующие макросы: `{HOST.CONN}`, `{HOST.DNS}`, `{HOST.ID}`, `{HOST.IP}`, `{HOST.HOST}`, `{HOST.NAME}` и пользовательские макросы `{$MACRO}`. Это очень удобно, так как с их помощью можно динамически генерировать адреса URL с целью извлечения данных из внешних источников.

Для правильного отображения динамических элементов адресов URL в разделе **Monitoring** ⇨ **Screens** (Мониторинг ⇨ Комплексные экраны) нужно выбрать хост. Если хост не выбран, вы увидите простое сообщение: «No host selected» («Не выбран узел сети»).

Динамические элементы можно идентифицировать установкой флажка, изображенного на рис. 5.32.

Теперь можно добавить в экран карту, например, или несколько динамических элементов, таких как графики. При добавлении динамического элемента в верхней части экрана появляется панель с раскрывающимися списками (рис. 5.33), которые помогут вам изменить цель динамического элемента.

Рис. 5.32 ❖ Флажок идентификации динамического элемента

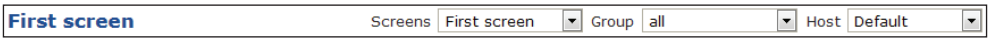


Рис. 5.33 ❖ Панель с дополнительными настройками для динамических элементов

На рис. 5.34 приводится пример комплексного экрана с динамическими и стандартными элементами.

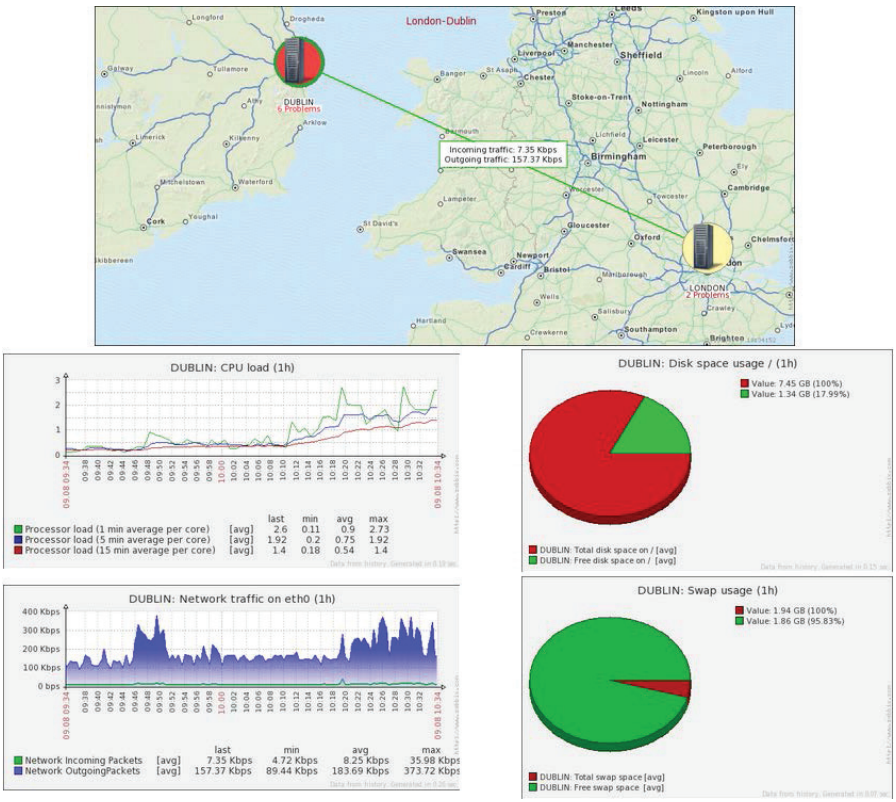


Рис. 5.34 ❖ Пример комплексного экрана

Очевидно, что выбор хоста в панели затронет только динамические графики. Вы можете таким способом переключаться между хостами и изменять ось *x*. Это заставит все динамические элементы обновиться соответственно.

Круговые и расширенные круговые диаграммы отображают только среднее значение за выбранный период. Для отображения связанных групп параметров лучше использовать пользовательские графики.

Слайд-шоу

После создания всех необходимых комплексных экранов можно организовать поочередный их показ на мониторе с помощью функции отображения слайд-шоу.

Создается слайд-шоу просто: перейдите на вкладку **Configuration** ⇨ **Slide shows** ⇨ **Create slide show** (Настройка ⇨ Слайд-шоу ⇨ Создать слайд-шоу). Перед вами откроется форма, изображенная на рис. 5.35.

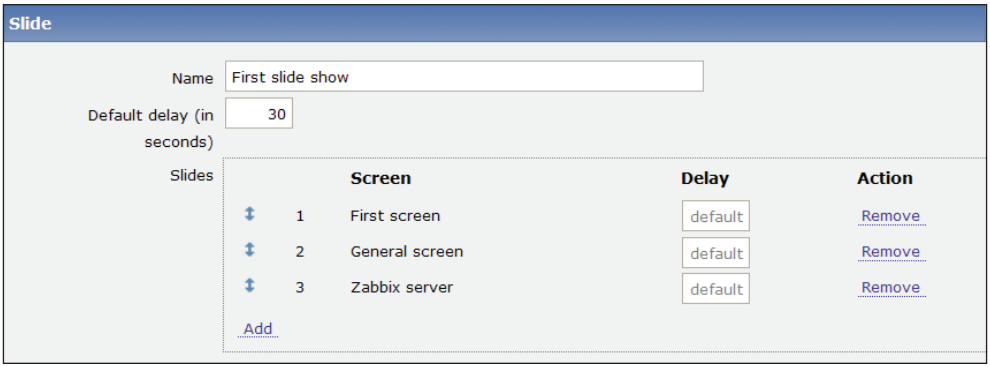


Рис. 5.35 ❖ Форма создания слайд-шоу

Как можно видеть на рис. 5.35, слайд-шоу имеет не очень много настроек, и все они интуитивно понятны. В поле **Name** (Имя) определяется имя слайд-шоу, а в поле **Default** (Задержка) – задержка между сменой слайдов (в секундах). Значение задержки по умолчанию действует для всех экранов, входящих в слайд-шоу.

В разделе **Slides** (Слайды), как можно видеть на рис. 5.35, перечисляются экраны в порядке отображения – вы можете изменить задержку для каждого из них. Таким способом фактически определяется время отображения каждого комплексного экрана. После сохранения слайд-шоу будет доступно в разделе **Monitoring** ⇨ **Screens** (Мониторинг ⇨ Комплексные экраны), где его можно выбрать по имени в раскрывающемся списке **Slide shows** (Слайд-шоу) справа.

В слайд-шоу можно добавлять только комплексные экраны. То есть, чтобы добавить единственный элемент, такой как график или карту, вам придется создать комплексный экран с этим элементом. Таким способом можно добавить в слайд-шоу все, что допускается использовать в комплексных экранах.

Если появится желание ускорить или замедлить смену экранов, это можно сделать, изменив множитель задержки. Для этого следует щелкнуть на пиктограмме меню (справа от раскрывающегося списка) и выбрать желаемый множитель, как показано на рис. 5.36.

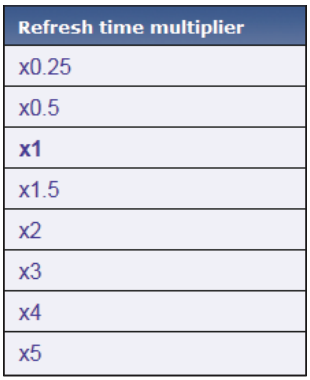


Рис. 5.36 ❖ Выбор множителя задержки

Проблема управления слайдами на большом мониторе

Отображение данных на большом мониторе представляет определенную проблему. Во-первых, необходимо знать свою целевую аудиторию, их уровень подготовки и какие функции они выполняют. Затем необходимо иметь полную информацию о том, где физически находятся мониторы, и их разрешении.

Для большого монитора вам, возможно, придется создать специальный экран, лучше умещающийся по его ширине. Экраны для широких мониторов должны использовать горизонтальное размещение элементов. Большинство комплексных экранов разрабатывается по аналогии с веб-страницами, поэтому их, вероятно, придется прокручивать вниз и вверх, чтобы увидеть все графики. Такие экраны плохо укладываются в широкие мониторы.

Учитывайте также тот факт, что слайд-шоу не выполняет автоматическую прокрутку вверх/вниз. Теоретически для этого можно добавить сценарии на JavaScript, но поверьте на слово, что реализовать экран, который правильно выполняет прокрутку, очень сложно, и все приложенные для этого усилия могут оказаться потраченными впустую. Гораздо проще и продуктивнее создавать экраны, четко укладывающиеся в размеры и разрешение монитора.

Замечания о слайдах для больших мониторов

Определив целевую аудиторию, их навыки и особенности работы, вы получаете точку опоры, благодаря которой сможете конструировать экраны для отображения на больших мониторах. В основном вам необходимо учитывать следующие факторы:

- простота представления (понятность);
- соответствие размерам большого монитора;
- неинтерактивность;
- задержка между сменой экранов.

Прежде всего старайтесь добиться максимальной простоты. Общее правило: чем проще представление, тем меньше усилий потребуется наблюдателю, чтобы воспринять информацию. Простота и понятность экрана уменьшают время реакции персонала. Предоставляйте информацию наиболее наглядным способом. Не перегружайте свои экраны избыточной информацией; вы должны найти правильный баланс между информативностью и простотой восприятия, и подберите хорошо читаемые шрифты. По сути, вам нужно определить, какая информация должна находиться на экране, и выбрать способы отображения подконтрольных служб.

Если требуется организовать мониторинг большого количества служб, выберите время задержки перед сменой экранов; не тратьте слишком много времени на отображение одного экрана, это раздражает наблюдателя, особенно если он ждет появления следующего слайда. К сожалению, в этом отношении нет определенного правила; вам придется потратить время на общение со службой поддержки, чтобы согласовать и выбрать наиболее подходящую задержку.

Не усложняйте себе работу, не старайтесь создавать сложные, многоуровневые экраны. При отображении на больших мониторах возможность переходить с уровня на уровень часто остается невостребованной. Работники будут только смотреть на большой монитор, а весь анализ – выполнять на своих настольных компьютерах.

Применение триггеров всегда идет только на пользу, так как они легко и быстро воспринимаются. Но не перегружайте ими страницу, так как это сделает ее нечитаемой.

Автоматизация слайд-шоу

После создания слайдов можно организовать их автоматическую смену. Думайте при этом о нуждах пользователя. У вас в системе наверняка имеется своя учетная запись.

На практике очень нежелательно отображать форму входа на больших мониторах. Чтобы избежать этого, создайте специальную учетную запись, имеющую некоторые ограничения.

Ваши главные цели:

- исключить возможность автоматического разъединения;
- избавиться от необходимости щелкать мышью для запуска слайд-шоу.

Если вам удастся достигнуть их, обслуживающий персонал будет только благодарен вам за это.

Чтобы исключить возможность автоматического разъединения, в форме настройки пользователя предусмотрен флаг **Auto-login** (Автовход). После установки этого флажка выполнить вход потребуется только один раз.

Флаг **Auto-login** (Автовход) действует, только если браузер поддерживает cookies, поэтому проверьте, не блокируются ли они какими-нибудь плагинами или антивирусами.

После создания специализированной учетной записи нужно настроить параметры в разделе **URL** (после входа); укажите здесь адрес URL вашего экрана.

Чтобы узнать соответствующий адрес URL, откройте слайд-шоу и скопируйте ссылку из адресной строки браузера. В данном примере эта ссылка имеет вид: `http://<zabbix-server> /zabbix/slides.php?sid=4258s278fa96eb&form_refresh=1&fullscreen=0&elementid=3`.

Фактически в этой ссылке достаточно оставить только параметр `elementid` и удалить параметр `sid`. Окончательная ссылка для данного примера имеет вид: `http://<zabbix-server>/zabbix/slides.php?form_refresh=1&fullscreen=0&elementid=3`.

Чтобы сразу перейти в полноэкранный режим, замените параметр `fullscreen=0` на `fullscreen=1`. Это поможет избавиться от необходимости дополнительного вмешательства со стороны человека.

Теперь для учетной записи настроена начальная страница. После входа автоматически запустится слайд-шоу в полноэкранном режиме.

Чтобы автоматизированное слайд-шоу отображалось правильно, не забудьте после запуска браузера перевести его в полноэкранный режим нажатием **F11**.

Информация об уровне обслуживания

Последний графический элемент, который мы обсудим в этой главе, – обзорный мониторинг инфраструктуры. Часто нас не интересуют низкоуровневые детали, нагрузка на процессор, потребление памяти или остаток свободного пространства на жестком диске. Гораздо больший интерес представляет факт доступности услуг, предоставляемых нашим вычислительным центром.

Zabbix охватывает эту область, предоставляя информацию об уровне обслуживания –иерархическое представление оказываемых услуг. Теперь представьте, что вам требуется организовать мониторинг веб-сайта (мы уже обсуждали уровни обслуживания в *главе 1* «*Развертывание Zabbix*»). Прежде всего нужно идентифицировать компоненты, обеспечивающие предоставление услуг, например веб-сервер, сервер приложений и сервер баз данных. Для каждого из них нужно определить триггеры, которые будут сообщать о доступности услуг. На рис. 5.37 приводится иерархическое представление услуги и ее компонентов.

Service	Status	Reason	Problem time	SLA / Acceptable SLA
root				
WEBSITE SLA Calculated	OK	-	0.0000	100.0000/ 99.9000
Web - Service on WEBSERVER is unavailable	OK	-	0.0000	100.0000/ 99.9000
RDBMS - Service on DATABASE SERVER is unavailable	OK	-	0.0000	100.0000/ 99.9000
Application - Service on DATABASE SERVER is unavailable	OK	-	0.0000	100.0000/ 99.9000

Рис. 5.37 ❖ Иерархическое представление услуги и ее компонентов

В этой иерархии каждый узел имеет состояние; это состояние вычисляется на основе триггеров и передается на уровень выше, в соответствии с выбранным алгоритмом. То есть на самом нижнем уровне состояние определяется триггерами.

Триггеры – это основа для информации об уровне обслуживания, поэтому они имеют по-настоящему большую важность. Выбирайте наиболее эффективные элементы для проверки этих триггеров.

Триггеры с уровнем важности **Information** (Информация) и **Not classified** (Не классифицировано) не должны рассматриваться и использоваться для вычисления оценки уровня обслуживания.

Настройка предоставления информации об уровне обслуживания

Для создания и настройки информационной службы, предоставляющей информацию об уровне обслуживания, перейдите на вкладку **Configuration ⇨ IT services** (Настройка ⇨ Услуги ИТ). На рис. 5.38 показана такая предварительно настроенная информационная служба:

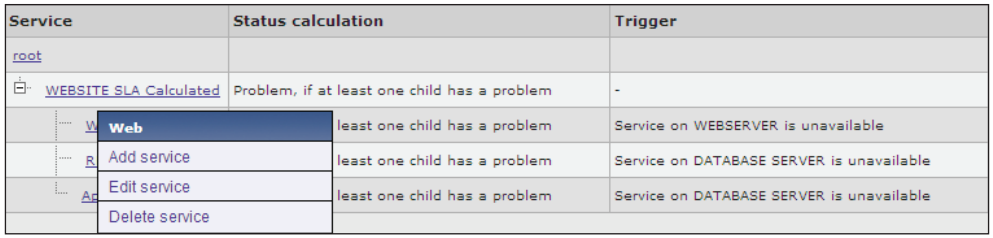


Рис. 5.38 ❖ Параметры настройки информационной службы

Щелкнув на ней, можно добавить новую, а также изменить или удалить существующую службу. Процедура настройки включает заполнение формы с тремя вкладками: на первой вводится описание службы, вторая вкладка предназначена для определения зависимостей, и третья – для ввода временных параметров.

На вкладке **Service** (Услуга) требуется ввести имя информационной службы. В данном конкретном примере выбрано имя «WEBSITE SLA Calculated»; конечно, веб-сайт состоит из нескольких разных компонентов, таких как веб-сервер, сервер приложений и СУБД. В трехуровневой архитектуре все они обычно развертываются на выделенных серверах. Теперь, поскольку каждый компонент играет важную роль в работе веб-сайта, нужно определить порядок вычисления оценки уровня обслуживания (SLA) для передачи уведомлений о проблеме. Это означает, что если на компоненте веб-сайта обнаружится проблема, будет считаться, что весь веб-сайт столкнулся с проблемой. Именно этот факт отражает настройка вычисления SLA.

Zabbix поддерживает три алгоритма вычисления состояния:

- **Do not calculate** (Без вычисления): этот алгоритм полностью игнорирует вычисление состояния службы;

- **Problem, if at least one child has a problem** (Проблема, если хотя бы одна дочерняя услуга в состоянии проблемы): если хотя бы один подчиненный компонент находится в состоянии «Проблема», вся служба считается недоступной. Это характерно для инфраструктур, в которых ни у одного из узлов нет горячего резерва;
- **Problem, if all the children has a problem** (Проблема, если все дочерние услуги в состоянии проблем): вся служба считается проблемной, если все подчиненные компоненты находятся в состоянии «Проблема». Это характерно для кластеров или служб, оснащенных балансировщиком нагрузки, когда имеется несколько узлов, обеспечивающих работоспособность службы, и все компоненты должны столкнуться с проблемой, чтобы родительский так же считался проблемным.

После определения алгоритма нужно указать процент SLA для вашей службы. Он используется для оценки важности возникающих проблем и отображения их цветом в отчетах.

Следующий шаг – определение триггера, который уведомит Zabbix о проблеме. Поскольку Zabbix поддерживает иерархическое представление, ваша основная служба может состоять из нескольких компонентов, соответственно, на промежуточных уровнях можно не определять триггеры, но они обязательно должны быть определены на самом нижнем уровне.

Последний параметр настройки: **Sort order** (Порядок сортировки). Он не влияет на вычисление SLA и носит исключительно косметический характер. В отчетах, например, ваши три уровня будут отсортированы в логическом порядке: веб-сервер, сервер приложений и сервер баз данных. Все вышесказанное отражает скриншот на рис. 5.39.

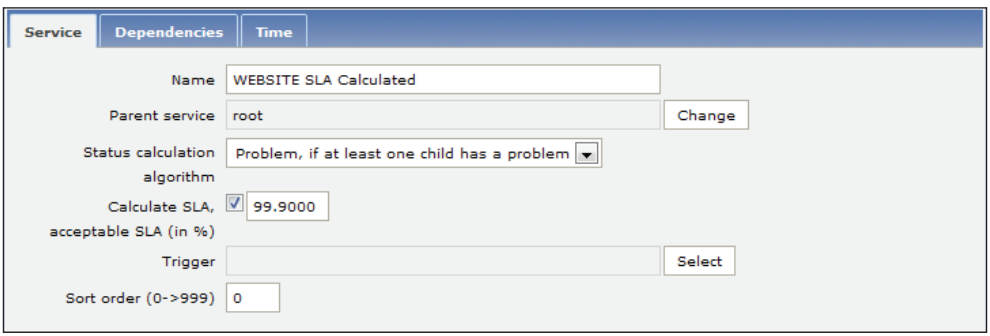


Рис. 5.39 ❖ Настройки информационной службы на вкладке **Service** (Услуга)

На рис. 5.40 изображены настройки зависимостей; в данном примере нет нужды определять каждую из зависимостей, поскольку их список составляется автоматически, в момент создания иерархического представления. Однако может так получиться, что одна из контролируемых служб была определена на другом уров-

не. В этом случае ее следует пометить как «нежесткую» зависимость, установив флажок **Soft** (Нежесткая).

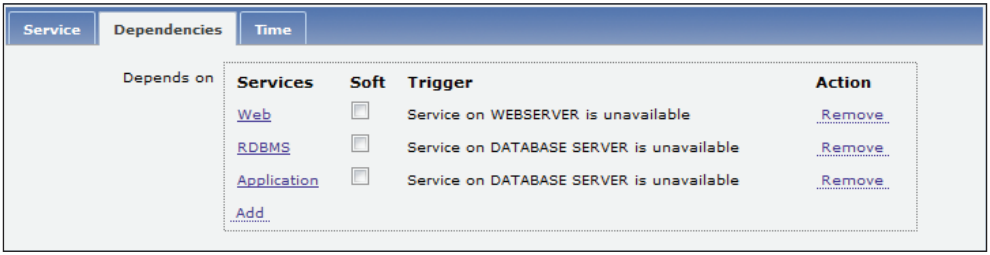


Рис. 5.40 ❖ Настройки зависимостей

Если информационная служба имеет только нежесткие зависимости, ее можно удалить, не удаляя перед этим всех подчиненных ей служб; этот прием можно использовать для быстрого удаления всей иерархии.

На третьей вкладке определяется время предоставления услуг. По умолчанию предполагается, что услуга предоставляется 24 часа в сутки, 7 дней в неделю, круглый год ($24 \times 7 \times 365$). К счастью для системных администраторов, не ко всем службам предъявляются такие требования. Если это так, можно определить свои периоды **Uptime** (Доступен) и **Downtime** (Недоступен), как показано на рис. 5.41.

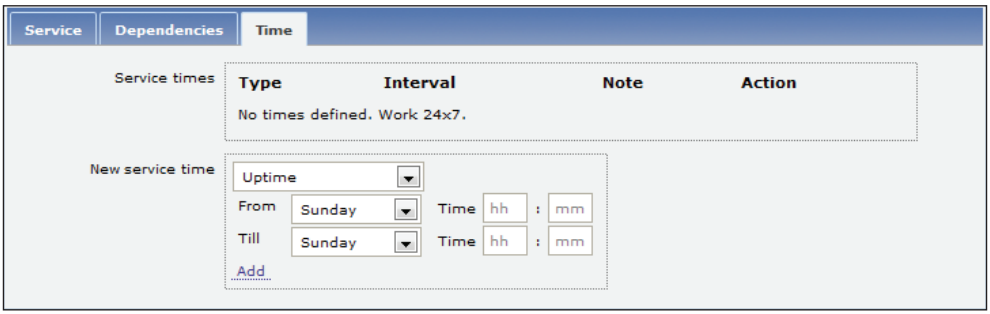


Рис. 5.41 ❖ Настройки времени предоставления услуг

Здесь определяются периоды доступности и недоступности службы. Проблемы, возникающие в период недоступности, не оказывают влияния на вычисление оценки уровня обслуживания (SLA). Здесь также можно добавить периоды однократного простоя, что может пригодиться для проведения запланированных профилактических работ без влияния на оценку SLA.

Закончив настройку иерархии предоставления информации об уровне обслуживания, результат можно наблюдать на вкладке **Monitoring ⇨ IT services** (Мониторинг ⇨ Услуги IT).

В заключение

В этой главе мы рассмотрели все графические элементы, поддерживаемые системой Zabbix, и как наиболее эффективно их использовать. В этой главе вы также узнали, как создать слайд-шоу для обслуживающего персонала, и познакомились с некоторыми эффективными приемами решения этой сложной задачи. Теперь вы должны хорошо понимать, что эта часть системы Zabbix требует определенного времени для реализации. Кроме того, она помогает упростить восприятие информации нетехническими специалистами, поскольку графическое представление информации людьми всегда воспринимается лучше. Решение данной задачи требует проявлять особую внимательность, но ваши усилия окупятся сторицей, и множество мощных графических элементов даст вам веские аргументы, когда это потребуется. Графические элементы помогут вам создать прочное обоснование, когда придется доказывать руководству необходимость расширения бизнеса или закупки новой техники.

В следующей главе вы увидите, как управлять сложными триггерами и их состояниями и как правильно выбрать количество триггеров и тревог для реализации, чтобы не перегрузить систему избыточными тревогами и не потерять особенно важных.

Глава 6

Управление оповещениями

Проверка условий и вывод предупреждений – одна из наиболее характерных функций систем мониторинга, и система Zabbix – не исключение. Но она имеет свою уникальную особенность: все условия, или триггеры (как они называются в этой системе), можно связывать не только с единственным измеряемым параметром, но и с вычислениями произвольной сложности, основанными на любых данных, доступных серверу Zabbix. Кроме того, так же как триггеры не зависят от элементов данных, действия сервера, предпринимаемые на основе состояний триггеров, не зависят от отдельных триггеров, в чем вы убедитесь, прочитав следующие разделы.

В этой главе вы узнаете следующие подробности о триггерах и действиях:

- создание сложных, интеллектуальных триггеров;
- минимизация вероятности ложных срабатываний;
- настройка автоматического выполнения операций на основе состояния триггера;
- эскалация действий.

Эффективная, правильная и всеобъемлющая настройка системы оповещений является ключом к успеху инфраструктуры мониторинга. Она основывается на обширной коллекции данных, как рассказывалось в *главе 4* «*Сбор данных*», и основной ее задачей является управление оповещениями, получателями и способами оповещения. Но все в этой системе крутится вокруг условий, объявленных для проверки, и в этом заключается главная цель триггеров.

Выражения триггеров

Триггеры – очень простые компоненты. Чтобы создать триггер (см. рис. 6.1), достаточно определить его имя и важность, а также простое выражение, определяющее условие срабатывания. Доступ к форме для ввода выражения предоставляется после щелчка на кнопке **Add** (Добавить), где вы сможете выбрать элемент, функцию для применения к элементу и определить некоторые другие параметры.

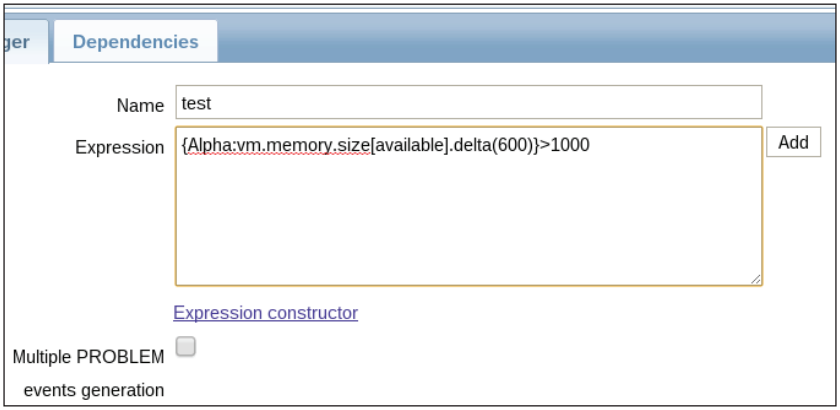


Рис. 6.1 ❖ Форма создания триггера

Как видите, здесь в выражении используется полная спецификация ключа элементов, а не просто его имя. Результат применения функции затем сравнивается с константой с помощью оператора «больше чем». Синтаксис ссылки на ключи элементов близко напоминает синтаксис определения вычисляемых элементов. Помимо ссылок на значения элементов, в триггерах можно использовать операторы сравнивания, которые возвращают логический результат. Они являются универсальными средствами унификации триггеров; не важно, насколько сложным будет выражение, – оно всегда должно возвращать `True` или `False`. Разумеется, это значение напрямую связано с состоянием триггера, которое может иметь два значения: `OK`, если выражение вернуло `False`, и `PROBLEM`, если выражение вернуло `True`. Триггеры не могут иметь никаких других состояний.

Триггер может также находиться в состоянии `UNKNOWN` (неизвестно), если его выражение не может быть вычислено (например, из-за отсутствия данных в элементах).

Выражение триггера состоит из двух основных компонентов:

- функций, применяемых к элементам данных;
- арифметических и логических операций, выполняемых над результатами, возвращаемыми функциями.

С точки зрения синтаксиса, *элементы и функции* должны заключаться в фигурные скобки, как можно видеть на рис. 6.1, а арифметические и логические операторы должны находиться за пределами фигурных скобок.

Выбор элементов и функций

В выражениях триггеров допускается использовать произвольное количество ссылок на элементы, при условии что к каждому элементу применяется своя функция. Это означает, что если вам понадобится использовать один и тот же элемент дважды, вы должны будете дважды использовать функцию, как показано ниже:

```
{Alpha:log[/tmp/operations.log,,,10,skip].nodata(600)}=1 or
{Alpha:log[/tmp/operations.log,,,10,skip].str(error)}=1
```

Триггер с этим выражением будет переходить в состояние PROBLEM, если в файле журнала operations.log в течение последних 10 минут не появилось ни одной новой строки, или если среди новых строк обнаружилось сообщение об ошибке.

Вычисления по короткой схеме для операторов and и or в Zabbix не реализованы (в версиях, предшествовавших Zabbix 2.4, эти операторы записывались как & и |); каждое сравнение вычисляется независимо от результатов предыдущих сравнений.

Разумеется, нет никаких ограничений, требующих использовать элементы только с одного хоста; в выражениях можно использовать любые элементы, получаемые с любых хостов и прокси (если к ним имеется доступ), как показано ниже:

```
{Proxy1:Alpha:agent.ping.last(0)}=0 and
{Proxy2:Beta:agent.ping.last(0)}=0
```

Триггер с этим выражением будет переходить в состояние PROBLEM, если оба хоста, Alpha и Beta, оказались недостижимы. Тот факт, что мониторинг этих хостов осуществляется двумя разными прокси, не имеет никакого значения. Все будет работать, пока прокси, указанные в выражении, будут иметь доступ к историческим данным двух подконтрольных хостов.

К элементам данных можно применять все функции, доступные для вычисляемых элементов. Полный перечень и описание функций можно найти в официальной документации Zabbix (https://www.zabbix.com/documentation/2.4/ru/manual/appendix/triggers/functions), поэтому я не буду повторять эту информацию здесь, но остановлюсь на некоторых аспектах, заслуживающих особого внимания.

Выбор между интервалом времени и количеством замеров

Многие функции, доступные в триггерах, принимают аргумент sec (количество секунд) или #num (количество замеров). Это означает, что можно указать период времени в секундах или количество замеров, и триггер будет извлекать все данные за указанный период и применять к ним функцию. То есть следующее выражение определит минимальное значение простоя процессора для хоста Alpha за последние 10 минут (600 секунд):

```
{Alpha:system.cpu.util[,idle].min(600)}
```

Следующий код, напротив, выполнит ту же операцию для десяти последних замеров:

```
{Alpha:system.cpu.util[,idle].min(#10)}
```

Вместо секунд можно использовать другие единицы измерения, такие как 10m (10 минут), 2d (2 суток) и 6h (6 часов).

Так какой же аргумент предпочтительнее, sec или #num? Совершенно очевидно, что ответ на этот вопрос зависит от целей и потребностей. Каждый из этих двух

аргументов имеет свои сильные стороны и должен использоваться в зависимости от контекста. Для всех пассивных проверок, инициируемых сервером, часто предпочтительнее использовать период времени, выраженный в абсолютных величинах. Параметр `#5`, например, может охватывать абсолютно несопоставимые периоды времени, когда интервалы проверок соответствующих элементов разнятся существенно. Хотя это и не так очевидно, но подобная подмена аргументов затронет также соответствующие триггеры. Кроме того, аргумент, выражающий период времени, может точнее соответствовать вашим целям, и вам проще будет понять определение триггера, когда вы вернетесь к нему спустя какое-то время. С другой стороны, для многих активных проверок предпочтительнее использовать аргумент `#num`, так как нет никаких гарантий, что между замерами будут выдерживаться постоянные интервалы времени. Это особенно верно для элементов-ловушек (трапперов) и проверок файлов журналов. При работе с такими элементами применение аргумента `#num` часто оказывается лучшим выбором.

Функции определения даты и времени

Все функции, возвращающие значение времени – текущее время, текущую дату, число месяца или день недели, – все еще требуют наличия допустимого элемента в составе выражения. Они могут пригодиться для создания триггеров, срабатывающих только в определенные часы суток или по определенным числам месяца, или, напротив, для определения периодов-исключений, когда возникают ожидаемые события, считающиеся ненормальными в другое время, например быстрое заполнение файловой системы файлами журналов из-за ошибки в некотором приложении. Команда разработчиков, работающая над устранением этой ошибки, могла бы обратиться к вам с просьбой понаблюдать за файловой системой и организовать остановку приложения, если оно начнет слишком быстро заполнять дисковое пространство журнальными записями. Как и многие другие, эту частную проблему можно решить в Zabbix несколькими способами, но вы, к примеру, решили не усложнять задачу и использовать в качестве индикатора увеличение объема используемого пространства в файловой системе более чем на 3% за последние 10 минут:

```
{Alpha:vfs.fs.size[/var,pused].delta(600)}>3
```

Единственная проблема этого выражения – оно не учитывает влияния законного процесса, который запускается каждую ночь в 2:00 и перемещает в эту же файловую систему пару огромных файлов. Это вполне обычная операция, но она может вызвать переход триггера в состояние `PROBLEM` и отправку оповещений. Эту проблему можно исправить, добавив пару функций для работы со временем:

```
{Alpha:vfs.fs.size[/var,pused].delta(600)}>3 and
({Alpha:vfs.fs.size[/var,pused].time(0)}<020000 or
 {Alpha:vfs.fs.size[/var,pused].time(0)}>030000 )
```

Но имейте в виду, что все функции, используемые в триггерах, возвращают числовые значения, включая дату и время, поэтому очень непросто выразить какие-то

замысловатые даты, такие как *первый вторник месяца* или *последний месяц года* (вместо последних 30 дней).

Важность триггера

Важность – это чуть больше, чем просто подпись, присоединяемая к триггеру. В веб-интерфейсе разные значения важности отображаются разными цветами, точно так же можно предусматривать разные действия в зависимости от важности, но у важности нет никакого другого применения в системе. Это означает, что важность триггера не изменяется с течением времени, не зависит от длительности пребывания триггера в состоянии `PROBLEM`, и вы не можете присвоить разные уровни важности разным порогам в том же самом триггере. Если действительно необходимо организовать отправку предупреждения, когда диск заполнится на 90%, и оповещение о критической ситуации, когда диск заполнится на 100%, вам придется создать два разных триггера с двумя разными порогами и уровнями важности. Возможно, это не самое лучшее решение, которое может привести к появлению предупреждений, которые будут проигнорированы и не обработаны, и последующему появлению оповещений о критической ситуации, когда уже слишком поздно и служба стала недоступной. Избыточные триггеры и избыточные сообщения увеличивают вероятность ошибки и потери важного сообщения на фоне менее важных.

Гораздо лучше четко оценить серьезность ситуации переполнения диска и создать только один триггер с обоснованным пределом и, возможно, обеспечить эскалацию действий, если есть опасение, что предупреждение затеряется среди множества других.

Выбор между абсолютными и относительными значениями

Рассматривая встроенные элементы агентов, можно заметить, что многие измеряемые параметры могут выражаться и в абсолютных значениях, и в относительных (в процентах). Часто имеет смысл следовать этому правилу при создании собственных элементов, так как это может пригодиться в будущем. Однако когда дело доходит до создания триггеров на их основе, это решение может оказаться особо ценным, особенно в задачах мониторинга доступного дискового пространства.

Размеры файловых систем и особенности заполнения дисков могут сильно отличаться для разных серверов, конфигураций, реализаций приложений и требований пользователей. Например, для небольшого диска объем свободного пространства 5% может оказаться слишком маленьким и заслуживающим отправки оповещения для принятия незамедлительных мер. А для дискового массива те же самые 5% могут соответствовать огромному объему, настолько большому, что от вас не требуется незамедлительных действий, а только лишь иметь в виду, что в будущем понадобится увеличить объем дискового пространства. Подобные размышления могут привести вас к мысли, что относительные величины не особенно полезны в подобных случаях и даже что такие триггеры, реагирующие на объем свободного дискового пространства, не имеет смысла включать в шаблоны, а луч-

ше настраивать их в каждом отдельном случае. В этом есть определенный смысл, особенно для важных файловых систем, но в больших окружениях такой подход быстро станет чересчур утомительным из-за необходимости контролировать сотни разных файловых систем.

В подобных ситуациях можно использовать функцию `delta`, которая поможет создавать триггеры, достаточно обобщенные, чтобы применить к самым разным файловым системам, и достаточно специализированные, чтобы получать оправданные предупреждения для каждой из них. Для критически важных дисков вам все еще придется определять узкоспециализированные триггеры, но этого не миновать в любом случае.

Одни и те же проценты могут иметь разную значимость для разных дисков, но сходные изменения процента доступного пространства для разных дисков обычно имеют одинаковую значимость: диск заполняется с такой скоростью, что в ближайшее время это может вызвать проблемы:

```
{Template_fs:vfs.fs.size[/,pfree].last(0)}<5 and
({Template_fs:vfs.fs.size[/,pfree].delta(1d)} or
{Template_fs:vfs.fs.size[/,pfree].last(0,1d) } > 0.5)
```

Этот триггер переключится в состояние `PROBLEM` не только, когда объем свободного пространства станет меньше 5%, но также если за последние 24 часа этот объем уменьшился более чем наполовину (обратите внимание на параметр сдвига во времени в последней функции). Это означает, что независимо от размера диска триггер сработает, если диск будет заполняться слишком быстро. Отметьте также, что с уменьшением свободного пространства процент будет соответствовать все меньшему и меньшему абсолютному значению, а это значит, что по мере заполнения диска вы все чаще и чаще будете получать уведомление.

Для подобных проверок относительные значения оказываются более гибкими и понятными, чем абсолютные, и более пригодными для использования в шаблонах. С другой стороны, абсолютные значения могут оказаться предпочтительнее, когда требуется создать специализированный триггер для очень специфической файловой системы.

Операции как способ связывания

Возможно, вы обратили внимание, что практически все интересные выражения триггеров основаны на логических операциях между двумя или более простыми выражениями. Естественно, это не единственный способ создания полезных триггеров. Многие простые проверки состояния элемента `agent.ping` помогут сэкономить массу времени, но Zabbix также поддерживает относительно простую возможность определения мощных проверок, которые в других системах требуют нетривиальной реализации. Рассмотрим несколько примеров относительно сложных триггеров.

Вернемся к функциям для работы с датами и временем. Допустим, что у нас имеется триггер, проверяющий количество активных сеансов в приложении и посылающий уведомление, если это число опускается слишком низко в определенные

часы, потому что известно, что в системе всегда имеется несколько автоматизированных процессов, создающих и использующих сеансы в этот период (например, с 10:30 до 12:30). В другие часы количество сеансов непредсказуемо или не играет большой роли, поэтому их количество следует продолжать проверять, но посылать предупреждения уже не нужно. Первая упрощенная версия триггера могла бы выглядеть так:

```
{Appserver:sessions.active[myapp].min(300)}<5 and
{Appserver:sessions.active[myapp].time(0)} > 103000 and
{Appserver:sessions.active[myapp].time(0) } < 123000
```

`sessions.active` может быть нестандартным сценарием, вычисляемым элементом или чем-то еще. Здесь это лишь метка, чтобы сделать пример более удобочитаемым, и не соответствует действительно существующему встроенному элементу.

Единственный недостаток этого триггера – если в этот период времени количество сеансов упадет ниже пяти и не увеличится до 12:30, триггер останется в состоянии `PROBLEM` до следующего дня. Это может превратиться в большую неприятность, если для этого триггера настроено множество действий с эскалацией, которые безосновательно будут продолжать выполняться в течение целых суток. Но даже если вы не настроили эскалацию действий, вы все равно получите отчет, отражающий, что событие продолжалось практически 24 часа, что само по себе неправильно. Даже если вы не испытываете проблем с отчетами, отображение ложного состояния `PROBLEM` будет вводить в заблуждение обслуживающий персонал и не позволит им сосредоточиться в полной мере на обнаружении настоящих проблем, а спустя время они могут перестать обращать внимание на этот конкретный триггер.

Одно из возможных решений – заставить триггер возвращать состояние `OK` в часы за указанным периодом, если он находится в состоянии `PROBLEM`:

```
({Appserver:sessions.active[myapp].min(300)}<5 and
 {Appserver:sessions.active[myapp].time(0)} > 103000 and
 {Appserver:sessions.active[myapp].time(0) } < 123000)) or
({TRIGGER.VALUE}=1 and
 {Appserver:sessions.active[myapp].min(300)}<0 and
({Appserver:sessions.active[myapp].time(0)} < 103000 or
 {Appserver:sessions.active[myapp].time(0) } > 123000))
```

Первые три строки идентичны строкам из предыдущего триггера. К ним добавились строки, выражающие следующие условия:

- триггер находится в состоянии `PROBLEM` (см. примечание ниже о макросе `TRIGGER.VALUE`);
- количество сеансов меньше нуля (это условие никогда не будет истинным);
- текущее время находится за границами установленного периода (последние две строки имеют смысл, противоположный определению периода во второй и третьей строках).

Макрос TRIGGER.VALUE разворачивается в текущее значение триггера, выраженное в виде числа. Значение 0 означает OK, 1 означает PROBLEM, а 2 означает UNKNOWN. Макрос можно использовать везде, где допускается применять функции элемента, и его требуется заключать в фигурные скобки. Как показано в предыдущем примере, этот макрос удобно использовать, когда требуется определить порог или условие, зависящее от текущего состояния триггера.

Условие «количество сеансов меньше нуля» гарантирует, что вне установленного периода все выражение будет возвращать false, если триггер находится в состоянии PROBLEM. Значение false соответствует состоянию триггера OK.

Здесь мы не только связали значение элемента с периодом времени, но также гарантировали, что событие не будет возникать в неустановленные периоды времени.

Еще один интересный способ конструирования триггеров – комбинирование разных элементов данных с одного хоста или даже разных элементов с разных хостов. Этот прием часто используется для выявления несоответствий в состоянии контролируемой системы, которые с трудом поддаются идентификации другими средствами.

Возьмем для примера сервер, занимающийся обработкой содержимого сети. Производительность такого сервера может сильно изменяться в зависимости от множества разных факторов, поэтому порой очень трудно определить обоснованные пороги для триггеров, чтобы уменьшить вероятность ложных срабатываний или, хуже того, пропуска событий. Однако можно с уверенностью утверждать, что если имеется высокая нагрузка на центральный процессор при небольшом сетевом трафике – это явный признак проблем:

```
{Alpha:system.cpu.load[all,avg5].last(0)} > 5 and
{Alpha:net.if.total[eth0].avg(300)} < 1000000
```

Другим хорошим примером может служить выявление зависших сеансов в приложении. Фактический способ обычно зависит от особенностей реализации приложения, но в данном примере будем предполагать, что компонент веб-интерфейса сохраняет множество временных файлов сеансов в определенном каталоге, а компонент базы данных заполняет таблицу информацией о сеансах. При наличии элементов данных, собираемых с этих двух хостов, каждое число само по себе, конечно, полезно для анализа тенденций и планирования модернизации, но, чтобы выявить несоответствия в работе приложения, эти данные должны сопоставляться друг с другом. Допустим, что в свое время мы реализовали локальную команду в агенте Zabbix, выполняющемся на стороне веб-интерфейса приложения, которая возвращает количество файлов в конкретном каталоге, и определили элемент ODBC на сервере баз данных, который извлекает из базы данных количество активных сеансов. На их основе можно построить триггер, сравнивающий два значения и переключающийся в состояние PROBLEM, если они не совпадают:

```
{Frontend:dir.count[/var/sessions].last(0)} <>
{Database:sessions.count.last(0)}
```

Пара символов <> – это оператор «не равно», который прежде записывался как #, а теперь, начиная с версии Zabbix 2.4, записывается как <>.

Агрегированные и вычисляемые элементы также могут пригодиться в создании эффективных триггеров. Следующий триггер проверяет, не опустилось ли слишком низко отношение количества работающих серверов в кластере к общему их количеству:

```
{ZbxMain:grpsum["grid", "proc.num[listener]", last, 0].last(0)} /
{ZbxMain:grpsum["grid", "agent.ping", last, 0].last(0)} < 0.5
```

Все эти примеры должны помочь донести до вас мысль, что как только вы выходите за рамки проверки простых порогов для отдельных элементов и начинаете увязывать разные источники данных в более сложные триггеры, перед вами открываются практически безграничные возможности определения новых триггеров.

Подбирая информативные параметры мониторинга, как описывалось в *главе 4* «*Сбор данных*», и объединяя их разными способами, можно точно определять различные аспекты поведения системы; можно сопоставлять записи о входе в систему в файлах журналов с активностью сети, для выявления возможных брешей в системе безопасности, сравнивать производительность отдельного сервера со средней производительностью серверов в той же группе для выявления проблем в доставке услуг и многое другое.

Фактически это одна из самых больших тайн Zabbix, которая заслуживает полного раскрытия. Система поддержки триггеров является сложным механизмом, опирающимся на простые и ясные методы конструирования выражений, а также на доступность обширной коллекции текущих и исторических данных. Время и силы, потраченные на изучение тонкостей и придумывание новых интересных и полезных триггеров, соответствующих вашим потребностям, окупятся сторицей, не только в виде эффективной и интеллектуальной системы мониторинга, но также в виде более глубокого понимания происходящего в вашем окружении.

Управление зависимостями триггеров

Очень часто доступность службы или хоста зависит не только от данного хоста, но и от доступности другого хоста, обеспечивающего возможность подключения. Например, если выйдет из строя маршрутизатор, недоступной окажется вся подсеть, и вы получите уведомления о недоступности всех хостов в подсети, даже притом, что проблема заключается в одном только маршрутизаторе. Определение отношения зависимости между маршрутизатором и хостами за ним могло бы помочь смягчить проблему за счет пропуска любых проверок в триггерах для хостов в подсети в случае недоступности маршрутизатора. Однако в Zabbix не поддерживаются зависимости между хостами, как в других системах, но поддерживаются зависимости между триггерами, что, по большому счету, суть одно и то же. Для каждого триггера можно определить другой триггер, от которого он будет зависеть. Если родительский триггер находится в состоянии `PROBLEM`, зависящий от

него триггер не будет проверяться, пока родительский не вернется в состояние OK. Этот подход не только обладает огромной гибкостью, но и имеет пару недостатков. Во-первых, для единственного хоста может быть определено существенное количество триггеров, поэтому, если потребуется определить зависимость между хостами, вы должны будете исправить каждый триггер для зависимого хоста, что может оказаться весьма непростой задачей. В подобных ситуациях иногда удается упростить проблему, добавив триггеры в собственный шаблон. Но в случае уникальности каждого хоста этот подход вряд ли поможет, потому что в результате придется создать шаблон для каждого уникального хоста, а это означает лишь, что утомительная работа просто переместится в шаблон. Впрочем, в некоторых случаях можно положиться на функцию массового обновления, поддерживаемую веб-интерфейсом Zabbix. Вторая проблема состоит в том, что, рассматривая определение хоста, нельзя сказать, зависит он от другого хоста или нет. Более того, беглого взгляда на триггеры хоста явно недостаточно, чтобы выявить этот вид отношений в Zabbix.

Важное преимущество поддержки зависимостей на уровне триггеров заключается в возможности определять зависимости между отдельными службами на разных хостах. Для примера представьте базу данных, обслуживающую несколько веб-приложений на разных веб-серверах. Если база данных окажется недоступной, ни один из зависящих от нее веб-сайтов не сможет работать, поэтому может оказаться желательным настроить зависимость между проверкой триггеров веб-серверов и доступностью базы данных. На тех же серверах может также иметься еще одна служба, работоспособность которой зависит от отдельного сервера лицензий или сервера идентификации и аутентификации. В таком случае можно определить соответствующие зависимости и получить в результате одну группу триггеров, зависящих от доступности одного сервера, и другую группу триггеров, зависящих от доступности другого сервера. Такие конфигурации сложно сопровождать, и все же небольшое их количество поможет избавиться от избыточных оповещений в больших окружениях. А это, в свою очередь, поможет сосредоточиться на настоящих проблемах, а не выискивать их в длинном потоке ложных оповещений.

Выполнение действий

По аналогии с элементами данных, которые просто предоставляют необработанные данные, триггеры, имеющие доступ к любым историческим данным, просто предоставляют доступ к изменениям своего состояния. Эти изменения фиксируются как события, по аналогии с тем, как результаты измерений записываются в элементы данных. Это означает, что триггеры не предоставляют никаких дополнительных функциональных возможностей – они просто проверяют условия и изменяют свое состояние по результатам проверки. И снова то, что кажется ограничением и недостатком, оказывается полной противоположностью: в Zabbix имеется компонент, фактически посылающий оповещения или пытающийся

автоматически решить некоторые проблемы независимым от триггера способом. Это означает, что так же, как триггеры имеют доступ к любым элементам данных, действия имеют доступ к любым триггерам, и на их основе можно создавать и универсальные, и узкоспециализированные действия, не ограниченные схемой «одно действие на триггер».

В отличие от триггеров, действия полностью не зависят от хостов и шаблонов. Каждое действие определяется глобально, а их условия проверяются с появлением любого события Zabbix. Как вы увидите далее, это часто вынуждает определять явные условия вместо неявных, но это компенсируется отсутствием необходимости создавать похожие, но разные действия для похожих событий просто потому, что они имеют отношение к разным хостам.

Всякое действие включает следующие три компонента, вместе обеспечивающие необходимую функциональность:

- определение;
- условия;
- операции.

Тот факт, что действие имеет глобальную область видимости, отражается на каждом его компоненте, но особенно это обстоятельство важно для условий, потому что именно условия определяют, какое действие должно быть выполнено в ответ на то или иное событие. Но давайте не будем забегать вперед и прежде познакомимся с интересными особенностями каждого компонента.

Определение действия

Под определением события подразумевается выбор имени действия и сообщения по умолчанию, отправка которого может входить в перечень выполняемых действий. В сообщении допускается ссылаться на конкретные данные, характеризующие событие, такие как имя хоста, элемента данных и триггера, значение элемента данных и триггера, а также адреса URL. Благодаря глобальности в определении события можно применять макросы, чтобы его можно было использовать в разных ситуациях и получать при этом полезную информацию.

На рис. 6.2 можно видеть несколько интересных макросов, участвующих в определении нового действия.

Назначение большинства очевидно, но обратите внимание, что здесь можно сослаться только на единственный триггер – триггер, вызвавший событие. С другой стороны, так же как триггер может проверять значения нескольких элементов данных с разных хостов, так же и в определении события можно ссылаться на любые элементы и хосты (до девяти разных хостов и/или событий), вовлеченные в событие, благодаря чему можно получить достаточно полную картину происходящего, просто прочитав сообщение.

Существует еще несколько макросов, способных увеличить информативность и выразительность сообщения. Но помните, что сообщение по умолчанию можно послать не только по электронной почте, но и в чат или в виде короткого сообщения SMS; почти наверняка у вас появится идея создать разные действия по

умолчанию с разными сообщениями для разных способов оповещения, и вам понадобится возможность регулировать объем передаваемой информации в том или ином случае.

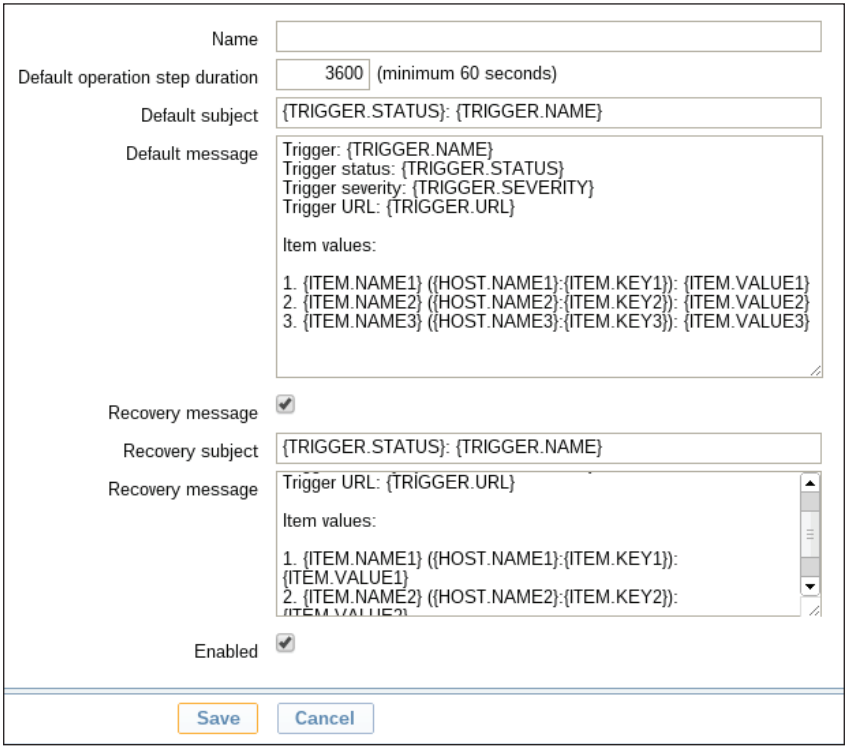


Рис. 6.2 ❖ Использование макросов в определении события

Полный список поддерживаемых макросов можно найти в официальной документации по адресу: https://www.zabbix.com/documentation/2.4/ru/manual/appendix/macros/supported_by_location, поэтому далее мы рассмотрим лишь наиболее интересные.

{EVENT.DATE} и {EVENT.TIME}

Эти два макроса помогут отделить время отправки сообщения от времени самого события. Это особенно полезно не только для повторной отправки или для эскалации действий, но также для всех способов оповещения, где сообщение не сопровождается отметкой времени.

{INVENTORY.SERIALNO.A} и подобные макросы

В случае с отказами аппаратного обеспечения информация о местоположении машины, контактная информация администратора, серийный номер и другие све-

дения могут оказаться очень полезными для быстрого поиска места аварии или вызова сторонней группы поддержки.

Определение условий

Этот компонент позволяет определять условия, опираясь на имя хоста, где произошло событие, а также имя триггера и его значение. Так же как в выражениях триггеров, здесь можно объединять разные простые условия с помощью логических операторов **AND**/**OR**, как показано на рис. 6.3. Можно использовать только операторы **AND**, только операторы **OR** или любые их комбинации.

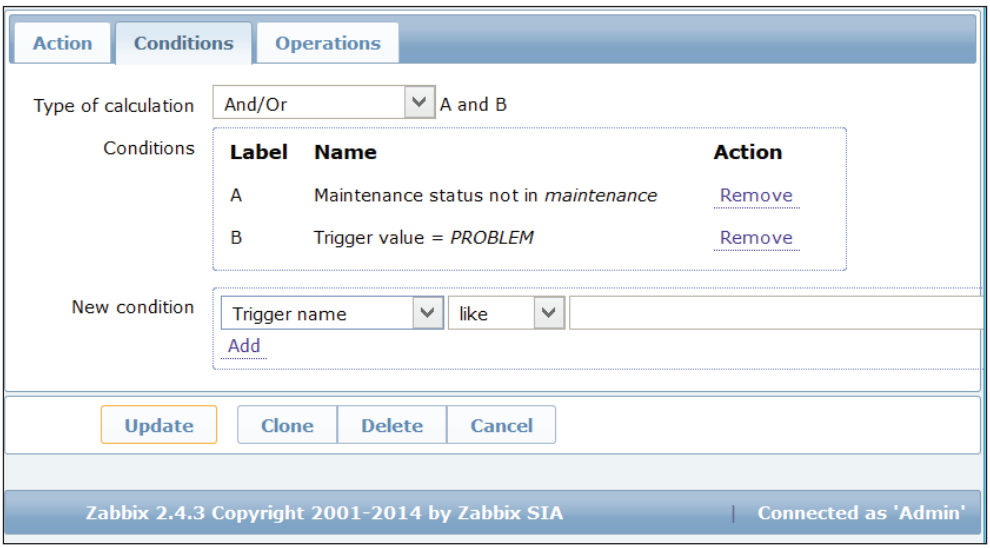


Рис. 6.3 ❖ Определение условий

Обратите внимание на условие `Trigger value = PROBLEM`. Так как условия вычисляются с каждым событием и переключение триггера между состояниями `PROBLEM` и `OK` само по себе является событием, если не указать это условие, действие будет выполняться и при переключении триггера в состояние `PROBLEM`, и при переключении в состояние `OK`. В зависимости от структуры сообщения по умолчанию и запрограммированных операций такое условие поможет реализовать ожидаемую реакцию Zabbix.

Если в форме определения действия вы добавите два разных сообщения и забудете включить условие, вы получите два сообщения: одно стандартное сообщение, в момент перехода триггера из состояния `OK` в состояние `PROBLEM`, и второе – сообщение о восстановлении работоспособности при переключении триггера обратно в состояние `OK`. Такое поведение может оказаться нежелательным. В принципе, нет ничего плохого в дублировании сообщения о восстановлении работоспособности, но если в действии запрограммировано обращение к внешним командам, такое

поведение может стать источником проблем. Если забыть указать условие `Trigger value = PROBLEM`, внешняя, удаленная команда будет вызвана дважды: в момент перехода триггера из состояния `OK` в состояние `PROBLEM` (как и предполагалось) и при переключении обратно в состояние `OK` (что почти всегда нежелательно). Чтобы обезопасить себя в отсутствие каких-то специфических требований, всегда желательно добавлять условие `Trigger value = PROBLEM` в каждое новое действие и проверять его наличие в действиях, которые вы собираетесь изменить.

Типичным примером создания разных действий с разными условиями является настройка отправки разным получателям предупреждений о неисправности и сообщений о восстановлении работоспособности. И это как раз та ситуация, когда следует помнить, что действия являются глобальными.

Допустим, что вы решили при любой неисправности баз данных посылать сообщение всем администраторам баз данных, а не администраторам Zabbix. Если просто создать новое условие, проверяющее принадлежность хостов к группе серверов баз данных, и в качестве получателей выбрать группу администраторов баз данных, они, конечно, получат все сообщения о событиях, связанных с базами данных, но точно так же эти сообщения получат администраторы Zabbix, если не настроить условия для действия по умолчанию. Так как действия являются глобальными, они всегда выполняются, когда их условия возвращают `True`. Если оба действия, специализированное и по умолчанию, определят истинность своих условий, обе группы получат сообщение. Чтобы избежать этого, нужно в действие по умолчанию добавить противоположное условие, которое было бы истинным для любого события, кроме тех, что входят в зону ответственности администраторов баз данных. Проблема в том, что такой подход может быстро выйти из-под контроля, и определение действия по умолчанию окажется битком набито условиями, отфильтровывающими сообщения. Поэтому, начиная создавать действия, специально предназначенные для узкой группы получателей, нужно либо отключить действие по умолчанию, либо использовать его исключительно для заполнения архива сообщений с целью последующего изучения администраторами.

Начиная с версии Zabbix 2.4 появился еще один способ вычисления условий. Как вы понимаете, тип вычислений **AND/OR** имеет множество ограничений. Возьмем для примера две группы с одинаковым типом условия: вы не сможете использовать условие **AND** в одной группе и условие **OR** – в другой. Начиная с версии Zabbix 2.4 это ограничение можно обойти. Если посмотреть список возможных вариантов вычисления условий, можно увидеть, что теперь появился вариант **Custom expression** (Пользовательское выражение), как показано на рис. 6.4.

Новый способ позволяет выполнять вычисления по собственным формулам, например:

- (*A и B*) *и* (*C или D*);
- (*A и B*) *или* (*C и D*).

И даже смешивать логические операторы, как в следующем примере:

- ((*A или B*) *и C*) *или D*.

Это открывает много новых интересных возможностей, позволяющих обходить ограничения, упомянутые выше.

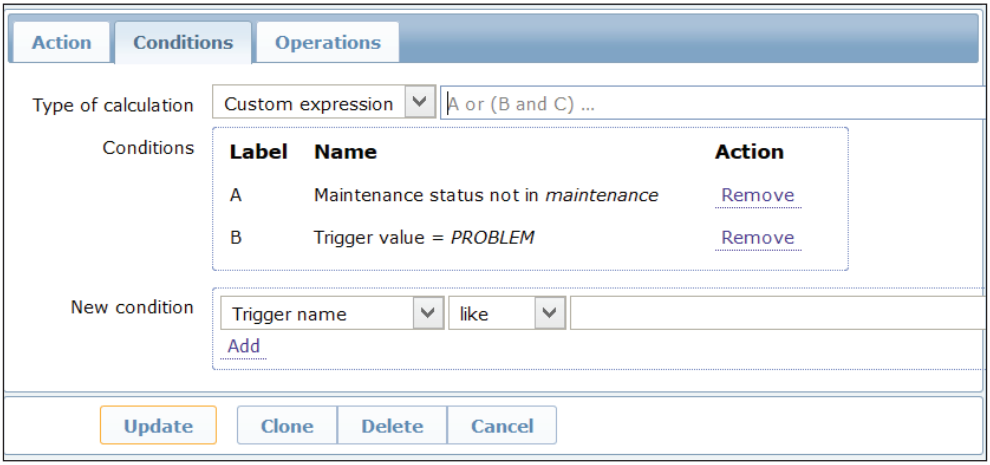


Рис. 6.4 ❖ Теперь появился вариант **Custom expression** (Пользовательское выражение)

Выбор операций

Если два предыдущих компонента являются лишь подготовительной частью, этот компонент определяет, какие операции должны фактически выполняться. Далее описываются два основных аспекта, связанные с этим:

- определение шагов;
- определение фактических операций в каждом шаге.

Так же как практически все остальное в Zabbix, самые простые случаи являются наиболее прямолинейными и очевидными, а именно: у вас имеется единственный шаг, включающий отправку сообщения по умолчанию определенной группе получателей. Кроме того, такой простой сценарий может усложняться все больше и больше, но все еще оставаться управляемым. Давайте познакомимся поближе с каждой его частью.

Шаги и эскалация

Даже если действие связано с единственным событием, это не означает, что оно может выполнять только одну операцию. В действительности оно может выполнять произвольное количество операций, называемых шагами, которые могут длиться неопределенное время или до момента, когда условия не станут ложными.

Для рассылки сообщений и выполнения автоматизированных операций можно использовать несколько шагов. Как вариант в разных шагах можно выполнять рассылку уведомлений разным группам или даже посылать его несколько раз одной и той же группе с некоторым интервалом, пока проблема не будет квитирована или устранена. Для примера на рис. 6.5 показана комбинация из нескольких шагов.

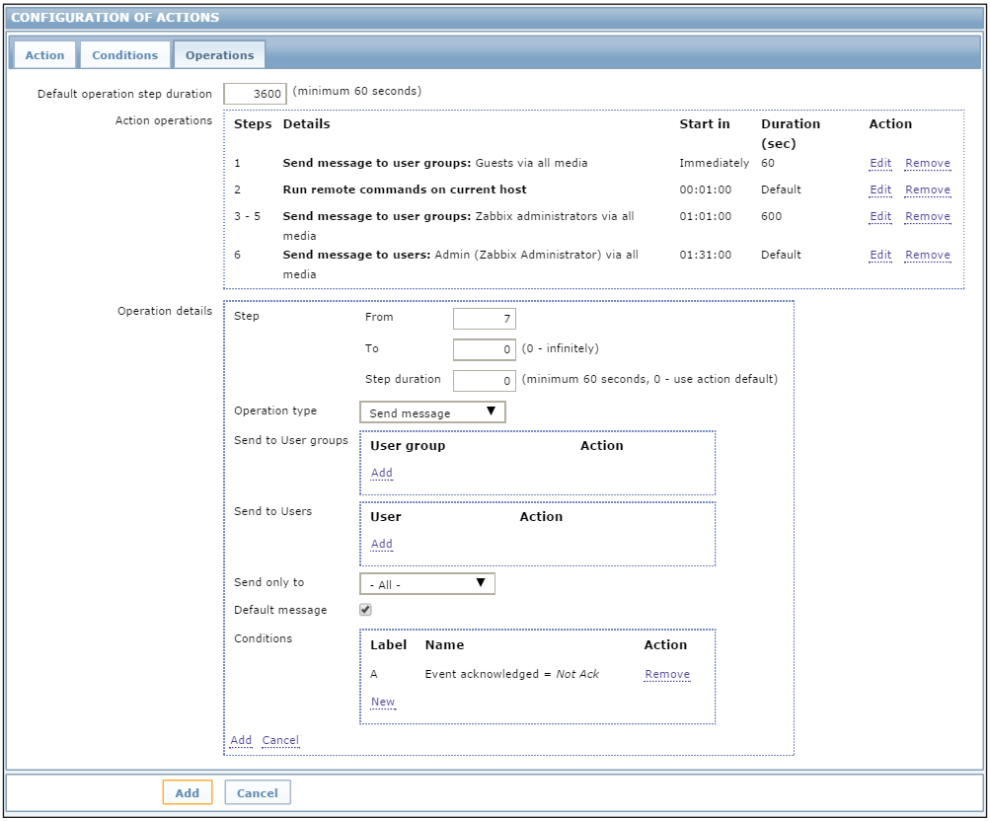


Рис. 6.5 ❖ Комбинация из нескольких шагов

Как видите, шаг 1 начинается немедленно (**Immediately**). Он отправляет сообщение группе пользователей и затем выполняет задержку перед следующим шагом на 1 минуту. Спустя минуту начинается шаг 2, который выполняет команду на удаленном хосте. Так как продолжительность (**Duration**) для шага 2 определена как **Default** (По умолчанию, выбирается на вкладке определения действия **Action** (Действие), в поле **Default operation step duration** (Длительность шага операции по умолчанию)), шаг 3 начнет выполнение примерно через час. Шаги 3, 4 и 5 идентичны друг другу и посылают сообщение группе пользователей с интервалом 10 минут, используя разные способы оповещения. Хотя на рис. 6.5 этого не видно, но шаг 6 выполняется, только если событие еще не было квитировано, а шаг 7 все еще находится в состоянии настройки. Интересная особенность шага 7 состоит в том, что он фактически должен объединить шаги с 7 по 0. Это может показаться странным, но в данном случае шаг 0 просто означает «бесконечно». Если в конфигурации действия предусмотрен шаг от *N* до 0, за ним не могут следовать никакие другие шаги, потому что такой шаг будет бесконечно выполняться с интервалом

времени, установленным в поле **Step Duration** (Длительность шага). Будьте внимательны, используя шаг 0, потому что такой шаг будет выполняться, пока состояние триггера не изменится. Более того, если вы не добавите условие `Trigger status="PROBLEM"`, шаг 0 продолжит выполняться и после того, как триггер вернется в состояние `OK`. Вообще, лучше никогда не использовать шаг 0, разве только в случаях, когда вы точно знаете и понимаете, что делаете.

Сообщения и способы оповещения

В каждом шаге, осуществляющем отправку сообщения, можно настроить отправку сообщения по умолчанию, указанного на вкладке **Action** (Действие), или другого сообщения, которое будет отправляться так же, как сообщение по умолчанию. Это может пригодиться, когда желательно добавить больше информации о событии в сообщение, посылаемое по электронной почте группе технических специалистов, или, напротив, сократить количество деталей в сообщении, посылаемом руководству, например, в виде SMS.

Помните, что в форме настройки операций для выбора доступны только получатели, являющиеся пользователями или группами Zabbix, при этом для них должны быть определены способы оповещения. Способы оповещения определяются на вкладке **Administration** (Администрирование) в веб-интерфейсе Zabbix для каждого пользователя в отдельности. Имейте также в виду, что каждый способ оповещения может быть разрешен или запрещен для того или иного пользователя; пользователь может быть доступен только в определенные часы или только для событий с определенным уровнем важности, как показано на рис. 6.6.

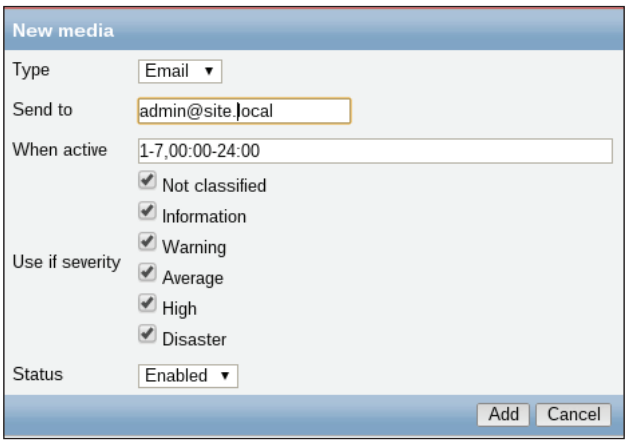


Рис. 6.6 ❖ Настройка способов оповещения сообщений для пользователя

Это означает, что даже если для действия настроена отправка сообщения, некоторые пользователи могут не получить его из-за настроек способов оповещения.

Даже притом, что **Email**, **Jabber** и **SMS** являются вариантами отправки, доступными по умолчанию, вам все еще нужно указать, как Zabbix должен посылать их. Это делается в разделе **Media types** (Способы оповещений) на вкладке **Administration** (Администрирование) в веб-интерфейсе. Имеется также возможность определять свои способы оповещения, которые будут доступны для выбора в формах настройки учетных записей пользователей и в настройках операций.

Если у вас есть несколько серверов и вы должны использовать их для разных целей или с разными идентификаторами отправителя, новым способом оповещения может быть другой сервер электронной почты, jabber или SMS. Это также может быть сценарий, что особенно интересно.

Сценарий, определяющий способ оповещения, должен находиться на сервере Zabbix, в каталоге, указанном в переменной `AlertScriptPath`, в файле `zabbix_server.conf`. Этот сценарий вызывается сервером с тремя параметрами:

- `$1`: адрес получателя сообщения;
- `$2`: тема сообщения;
- `$3`: тело сообщения.

Адрес получателя извлекается из соответствующего свойства пользователя, которое вы определили при создании нового способа оповещения. Тема и тело сообщения настраиваются в определении действия или шага, как было описано выше. Затем, как предполагает сервер Zabbix, независимо от адреса получателя, будь то ссылка UUCP, современный почтовый сервер, требующий строгой аутентификации, или внутренний сервер микроблогинга, сценарий должен послать сообщение запрограммированным в нем способом. Фактически вы можете сделать с сообщением все, что угодно: зарегистрировать его в журнале, отправить на удаленный файл-сервер, превратить его в запись syslog и послать на сервер журналирования, вызвать программу синтезатора речи и вывести сообщение на динамики или записать на автоответчик; ваши возможности ограничиваются только вашей фантазией. Но не путайте нестандартные способы оповещения с удаленными командами – и тот, и другой механизмы позволяют получать одни и те же результаты, но в действительности это совершенно разные вещи.

Удаленные команды

Удаленные команды обычно используются для выполнения корректирующих действий для исправления проблемы без участия человека. После выбора целевого хоста, который должен выполнить команду, сервер Zabbix соединится с ним и отдаст указанную команду. Если для взаимодействий используется агент Zabbix, в настройках агента присвойте параметру `EnableRemoteCommands` значение `1`, иначе агент никаких команд выполнять не будет. Еще один способ выполнения удаленных команд – через SSH, Telnet и IPMI (если вы включили соответствующую поддержку при компиляции сервера, во время его установки).

Удаленные команды можно использовать практически для всего, чего угодно, – для остановки или перезапуска процессов, для освобождения дискового пространства путем архивирования или удаления старых файлов, для перезагрузки

машины и т. д. Начинающим администраторам они кажутся мощным, потрясающим средством, но по своему опыту могу сказать, что это чрезвычайно хрупкое решение, способное порождать новые проблемы ничуть не реже, чем исправлять существующие. Очень сложно обеспечить полную безопасность их выполнения и избежать удаления файлов или перезагрузки серверов по ошибке. Проблема удаленных команд состоит в том, что они чаще маскируют проблемы, а не делают их явными, в чем, собственно, и заключается главная задача системы мониторинга. Да, они могут оказаться действенным средством быстрого исправления каких-то проблем и обеспечения безотказной работы службы, но если вы будете злоупотреблять ими, вы быстро забудете, что проблемы имеют свойство возвращаться, если не проявлять к ним должного внимания. Обычно намного лучше попытаться решить проблему по-настоящему, чем скрывать ее за временной ширмой автоматически выполняемых команд. Это не просто философское заключение – команды, терпящие неудачу, могут привести к весьма пагубным последствиям.

Поэтому примите совет: используйте удаленные команды разумно, и только если вы знаете, что делаете.

В заключение

В этой главе рассказывалось обо всем, что обычно считается основой систем мониторинга, – средствах инициации операций и рассылки оповещений. После поочередного знакомства с двумя компонентами этой функции – триггерами и действиями – вам должно быть понятно, как философия разделения функций, исповедуемая в Zabbix, помогает получить массу выгод. Здесь вы узнали, как создавать сложные условия для триггеров, помогающие обеспечить более точное управление уведомлениями. Теперь многие функции и параметры триггеров, а также некоторые замечательные их особенности, наряду со многими аспектами создания действий, перестали быть для вас секретом.

В следующей главе мы займемся исследованием последней составляющей ядра Zabbix: шаблонов и функций обнаружения.

Глава 7

Управление шаблонами

При всех невероятных возможностях, заключенных в элементах данных, графиках, картах и триггерах Zabbix, было бы невероятно сложно вручную создавать каждый отдельный компонент для каждого подконтрольного хоста. В больших окружениях, с сотнями и тысячами объектов мониторинга, практически невозможно вручную настроить все необходимые элементы данных, графики и триггеры.

Механизм поддержки шаблонов дает возможность определить разные коллекции элементов, триггеров и графиков для применения к любому количеству хостов, сохраняя при этом возможность управлять отдельными аспектами, которые может потребоваться изменить в каждом отдельном случае.

Отличным дополнением к шаблонам является функция обнаружения. С ее помощью можно определить набор правил, чтобы Zabbix мог обнаруживать появление новых хостов без необходимости вручную настраивать их мониторинг. Также можно воспользоваться преимуществами функции низкоуровневого обнаружения, реализованной в агентах Zabbix и позволяющей автоматически привязывать необходимые элементы данных даже к часто изменяемым компонентам системы, таким как количество и разновидности дисков, файловых систем и сетевых интерфейсов.

В этой главе вы научитесь:

- создавать и использовать вложенные шаблоны;
- объединять несколько шаблонов для мониторинга хостов;
- использовать функцию обнаружения хостов и действия для назначения шаблонов новым хостам;
- настраивать низкоуровневое обнаружение, чтобы сделать шаблоны еще более универсальными.

Начнем с самого начала и посмотрим, чем отличается определение шаблона от обычного определения хоста, даже притом, что они выглядят одинаково.

Создание шаблонов

Определение шаблона хоста очень напоминает обычное определение хоста. Оба являются коллекциями элементов, триггеров, графиков, экранов и правил низкоуровневого обнаружения. Оба должны иметь уникальное имя, как и любые другие

сущности в системе мониторинга Zabbix. Оба могут принадлежать одной или нескольким группам. Главное отличие: для хоста определяется один или более способов взаимодействий, чтобы сервер Zabbix мог получать информацию из элементов данных, как описывалось в *главе 4 «Сбор данных»*. Это может быть один или несколько IP-адресов или имен хостов, представляющих интерфейс с агентом, адрес SNMP, JMX или IPMI. То есть хост – это объект, к которому сервер Zabbix может обратиться за информацией или получать данные по его инициативе. Шаблон, напротив, не имеет интерфейса для взаимодействий, поэтому сервер Zabbix никогда не пытается проверить работоспособность шаблона или запросить у него последние данные.

Создаются шаблоны просто, процедура создания практически не требует пояснений: достаточно перейти на вкладку **Configuration** ⇨ **Templates** (Настройка ⇨ Шаблоны) и щелкнуть на кнопке **Create template** (Создать шаблон). В результате откроется форма настройки нового шаблона с тремя вкладками. Вкладки **Linked templates** (Присоединенные шаблоны) и **Macros** (Макросы) мы рассмотрим ниже в этой главе, так как их заполнение не требуется для создания простого шаблона. В действительности единственным важным элементом простого шаблона является его имя, тем не менее будет полезно связать шаблон с одной или несколькими группами, чтобы потом его проще было отыскать в других разделах веб-интерфейса. Если у вас уже имеются настроенные хосты, вы можете связать шаблон с требуемыми хостами прямо на вкладке создания шаблона. В противном случае вам придется перейти на вкладку **Hosts** (Узлы сети) и привязать шаблон там. После создания шаблона он станет доступен в списке шаблонов, но все еще будет оставаться пустым объектом. Следующая ваша задача – добавить в шаблон элементы данных, триггеры, графики, экраны и правила обнаружения.

Добавление сущностей в шаблон

Добавление элемента или любого другого компонента в шаблон осуществляется практически так же, как добавление того же компонента в определение обычного хоста. Это особенно верно для элементов данных. Как вы уже знаете, ключи элементов являются основными строительными блоками конвейера мониторинга в системе Zabbix, и они не требуют указывать какие-либо адреса или интерфейсы, потому что всю эту информацию Zabbix получает из определения хоста. Это означает, что, добавляя элементы в шаблон, вы фактически создаете элементы для идеального хоста, которые позднее будут применяться к реальным хостам, подлежащим мониторингу.

Шаблоны, так же как хосты, по сути, являются коллекциями элементов данных, триггеров и графиков. Поскольку многие идеи, обсуждаемые далее, в равной степени применимы к элементам данных, триггерам и графикам, для ссылки на объекты этих трех типов в оставшейся части главы будет использоваться термин «сущность». Иными словами, увидев далее в тексте слово «сущность», вы можете мысленно заменять его словом «элемент данных», «триггер» или «график».

Все то же относится к другим типам сущностей, но, так как они всегда ссылаются на один или несколько элементов данных, вы должны убедиться, что эти элементы принадлежат шаблону, а не обычному хосту, как показано на рис. 7.1.

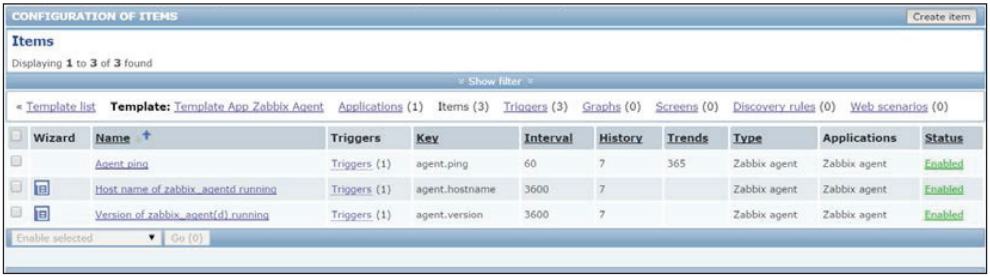


Рис. 7.1 ❖ Элементы должны принадлежать шаблону, а не обычному хосту

Хоть это и очевидно, но подмечу, что элементы, графики и экраны, содержащиеся в шаблоне, очень легко выбрать, пользуясь соответствующими ссылками в верхней части окна.

Главное отличие сущностей в шаблонах от сущностей в определениях хостов, особенно ярко проявляющееся, когда дело доходит до триггеров, связано с тем, что макросы позволяют сделать имена триггеров, графиков или параметров более выразительными и универсальными.

Ниже перечислены сущности, которые можно группировать в шаблоны:

- элементы данных;
- триггеры;
- графики;
- приложения;
- комплексные экраны;
- правила низкоуровневого обнаружения;
- веб-сценарии (начиная с версии Zabbix 2.2).

Важно также отметить, что для связывания шаблона с хостом сам хост должен иметь элементы с уникальными именами. То есть если хост уже содержит элементы, имена которых совпадают с именами элементов из шаблона, нужно устранить проблему дублирования.

Использование макросов

Как вы уже видели в *главе 6* «*Управление оповещениями*», макросы позволяют создавать сообщения, достаточно универсальные, чтобы их можно было применять ко множеству событий. Сервер Zabbix берет на себя труд развернуть все макросы в сообщении и заместить их фактическими данными, опираясь на конкретное событие. Поскольку сообщение в действии фактически является шаблоном, применяемым к конкретному событию, нетрудно понять, как фактически та же самая идея применяется к шаблонам хостов. Единственное, что меняется, – это

контекст; событие имеет весьма обширный контекст, позволяющий ссылаться на триггер и один или несколько разных элементов и хостов, а контекст простого хоста существенно уже. Это отражается на количестве доступных макросов, которые перечислены в табл. 7.1.

Таблица 7.1 ❖ Макросы, доступные в шаблонах хостов

Макрос	Транслируется в	Примечания
{HOST.CONN}	Имя хоста или IP-адрес	Действует идентично макросам {HOST.IP} и {HOST.DNS} в зависимости от параметра настройки **Connect to** (Подключаться через) в настройках хоста
{HOST.DNS}	Имя хоста	Полное имя хоста, как определено в системе доменных имен DNS
{HOST.HOST}	Имя хоста, как определено в Zabbix	Идентификатор хоста. Должен быть уникальным для конкретного сервера Zabbix. Если мониторинг хоста осуществляется с помощью агента, это же имя должно быть указано в его конфигурации
{HOST.IP}	IP-адрес хоста	Хост может иметь несколько IP-адресов. На них можно ссылаться с помощью макросов {HOST.IP1}, {HOST.IP2} и т. д., до {HOST.IP9}
{HOST.NAME}	Видимое имя хоста, как определено в Zabbix	Имя хоста, отображаемое в списках, картах, экранах и т. д.

Чтобы лучше понять, чем отличаются разные макросы {HOST.*}, рассмотрим пример конфигурации хоста, изображенной на рис. 7.2.

В данном случае макрос {HOST.HOST} будет замещаться именем ZBX Main, макрос {HOST.NAME} – именем Main Zabbix Server, {HOST.IP} – 127.0.0.1 и {HOST.DNS} – zabbix.example.com. Наконец, поскольку параметр **Connect to** (Подключаться через) имеет значение **IP**, макрос {HOST.CONN} будет замещаться IP-адресом 127.0.0.1.

Наиболее очевидная область применения этих макросов – определения триггеров и графиков. Так как имя графика выводится в его заголовке, очень важно отличать графики одного и того же вида, принадлежащие разным хостам, особенно когда они отображаются в одном комплексном экране, как описывалось в *главе 5* «*Визуализация данных*».

Менее очевидное применение этих макросов – в определениях ключей элементов. Мы уже затрагивали тему внешних сценариев в *главе 4* «*Сбор данных*», и вы еще раз столкнетесь с ними в следующей главе, поэтому сейчас я не буду сильно углубляться в их детали, а просто отмечу, что с точки зрения создания элемента достаточно знать лишь имя сценария и параметры, необходимые ему для корректной работы.

Поскольку внешние сценарии по своей природе не имеют доступа к информации в Zabbix, кроме входных параметров, часто бывает необходимо передавать им в одном из аргументов IP-адрес или имя хоста. Это гарантирует, что сценарий подключится к требуемому хосту и соберет необходимые данные. Единственный, правильно настроенный сценарий сможет выполнять одну и ту же операцию с любым количеством разных хостов благодаря системе шаблонов и макросам, таким как {HOST.CONN}, {HOST.IP} и др.

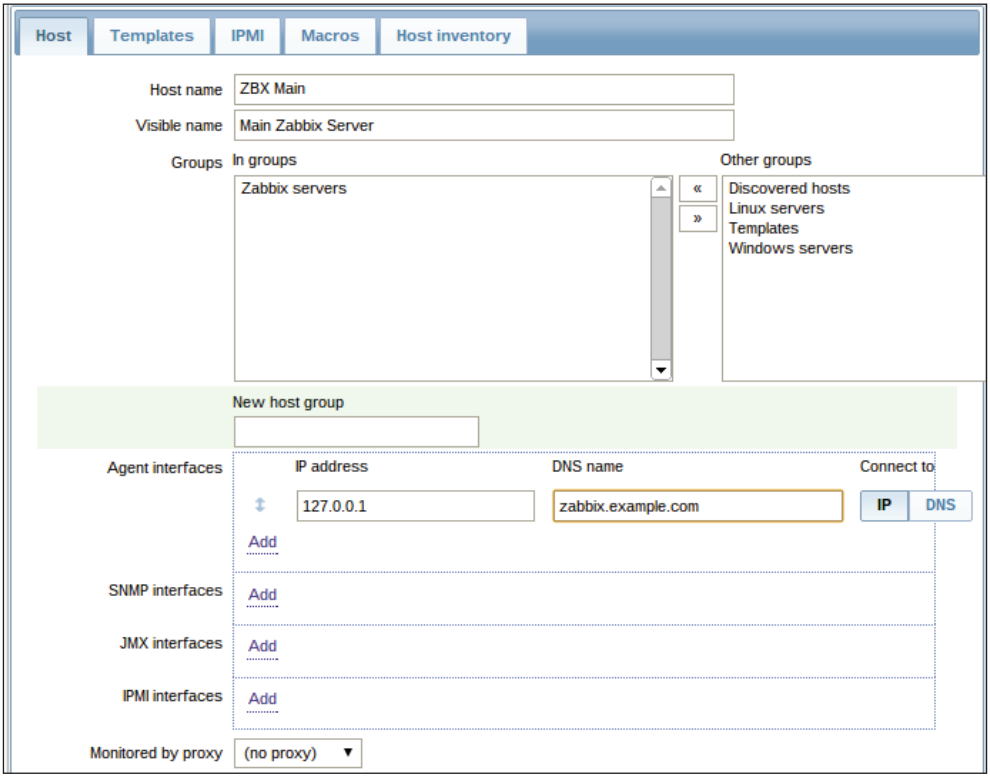


Рис. 7.2 ❖ Пример конфигурации хоста

Возьмем для примера сценарий, проверяющий работоспособность некоторого приложения с использованием нестандартного протокола. Допустим, что этот сценарий имеет имя `app_check.sh`, принимает имя хоста или IP-адрес в виде аргумента, соединяется с приложением, используя нестандартный протокол, и возвращает `1`, если приложение выполняется, и `0`, если проверка потерпела неудачу. В этом случае ключ соответствующего элемента данных мог бы выглядеть, как показано на рис. 7.3.

В таких случаях применение макроса в определении ключа элемента данных – единственный способ сделать внешнюю проверку частью шаблона.

В качестве еще одного примера рассмотрим группу хостов Zabbix, которые представляют не обычные машины с операционными системами, физические или виртуальные, а отдельные приложения или экземпляры баз данных. В такой ситуации все хосты-приложения могут иметь общий адрес и интерфейс – принадлежащие фактическому серверу, на котором выполняются приложения. Их можно связать с шаблоном, только если он имеет элементы данных уровня приложения (или уровня базы данных). Для простоты допустим, что у нас имеется сервер приложений (Alpha), на котором выполняются три приложения:

- архиватор документов (doku);
- форма опроса клиентов (polls);
- внутренний сайт микроблогинга (ublog).

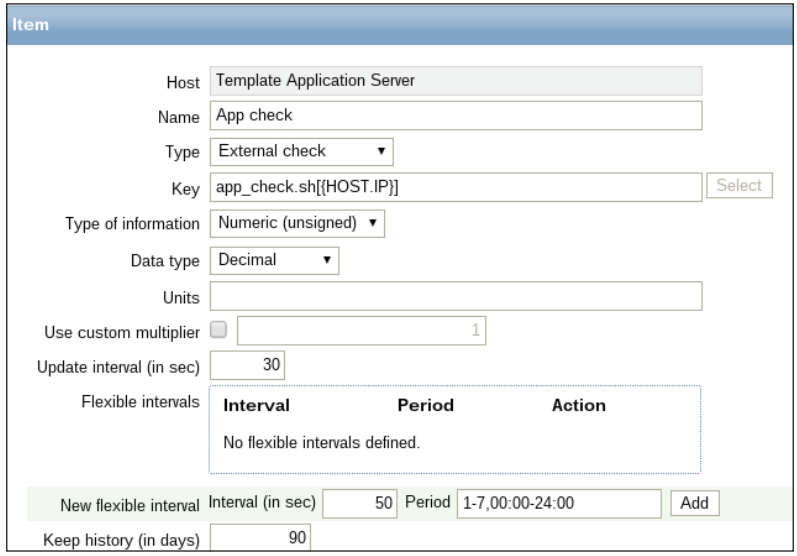


Рис. 7.3 ❖ Использование макроса в определении имени элемента данных

Нас интересуют следующие параметры работы каждого приложения:

- количество активных сеансов;
- объем потребляемой памяти;
- количество потоков выполнения;
- объем сетевого трафика;
- количество соединений с базой данных.

И снова для простоты допустим, что у нас имеется пакет внешних сценариев, которые по заданному IP-адресу и имени приложения способны извлекать перечисленные параметры. Ключи внешних сценариев обычно имеют простой для чтения вид, но ту же идею можно применить к значениям в консоли JMX, счетчикам производительности Windows, запросам к базе данных и любым другим элементам данных.

Один из способов реализовать мониторинг такой конфигурации – создать единственный хост Alpha и вдобавок ко всем обычным элементам мониторинга ОС и аппаратуры определить несколько элементов для мониторинга приложения. Такая связка, конечно, будет работать, но если добавится новое приложение, вам придется определить для него все необходимые элементы, триггеры и графики, несмотря на то что они будут отличаться от уже имеющихся только именем приложения.

Поскольку имя приложения – единственное, чем отличаются коллекции сущностей, есть более гибкое решение, которое заключается в том, чтобы оформить мониторинг каждого приложения в отдельное определение хоста и применить общий шаблон.

С точки зрения Zabbix, хост – это всего лишь коллекция сущностей с одним или несколькими интерфейсами для подключения. Хост не обязан быть фактической машиной, физической или виртуальной, со своей операционной системой. Роль хоста в Zabbix может играть любая абстрактная, но согласованная коллекция измеряемых параметров, имеющая некоторый способ подключения. Типичными примерами являются: приложения, экземпляры баз данных и т. д.

Вместо создания множества одинаковых элементов данных, триггеров и других сущностей для хоста Alpha можно создать собственный шаблон приложения и заполнить его требуемыми элементами, как показано на рис. 7.4.

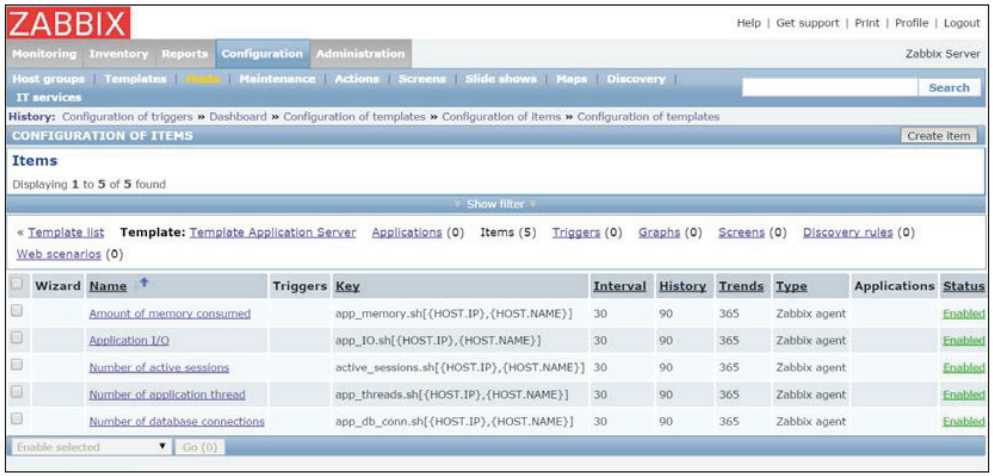


Рис. 7.4 ❖ Шаблон приложения

То есть для каждого приложения можно создать один хост с IP-адресом сервера Alpha и именем приложения в качестве имени хоста. Подключение такого шаблона к хостам позволит вам получать те же результаты, но более гибким способом; теперь, чтобы добавить мониторинг нового приложения, достаточно просто создать хост и связать его с требуемым шаблоном. Если впоследствии приложение будет перенесено на другой сервер, вам понадобится лишь изменить IP-адрес. Если поместите все такие хосты-приложения в отдельную группу, вы сможете даже дать право на доступ к измеряемым параметрам приложений определенной группе пользователей, не открывая для них доступа к данным мониторинга самого сервера приложений. И разумеется, добавление, удаление или изменение сущностей в шаблоне немедленно будет отражаться на всех контролируемых приложениях.

Пользовательские макросы

В Zabbix поддерживается особый класс макросов – пользовательские макросы уровня шаблона и хоста. Их можно определить на вкладке **Macros** (Макросы) в форме определения любого хоста или шаблона, а также в форме администрирования. Определяются макросы очень просто, так как они всего лишь задают перевод метки (имени макроса) в предопределенное, фиксированное значение, как показано на рис. 7.5.

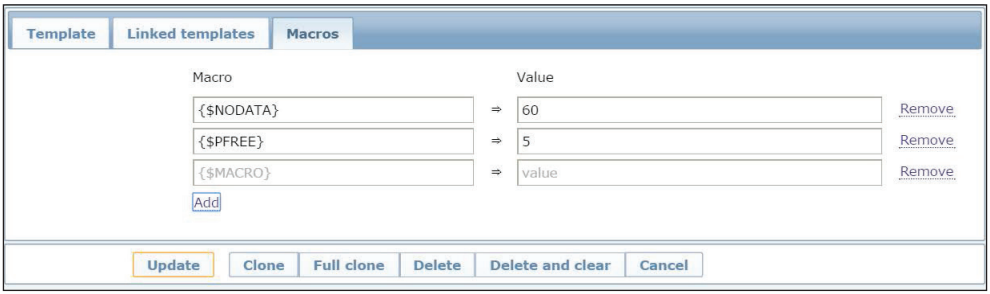


Рис. 7.5 ❖ Определение пользовательских макросов

В шаблонах они особенно полезны для определения пороговых значений триггеров – если впоследствии понадобится изменить несколько триггеров, достаточно просто изменить макрос `{$NODATA}`, а не править каждый отдельный триггер, использующий его. Пользовательские макросы можно использовать везде, где допускается использовать встроенные макросы.

Если пользовательский макрос используется в триггерах или элементах данных, объявленных в шаблоне, макрос предпочтительнее добавлять в шаблон. При таком подходе можно экспортировать шаблон в формат XML и импортировать в другую систему мониторинга, не заботясь об определении необходимых пользовательских макросов.

Польза от макросов еще выше, когда они используются во вложенных шаблонах, в чем вы вскоре убедитесь.

Глобальные макросы и макросы на уровне хоста обычно используются:

- чтобы получить все преимущества применения шаблона к атрибутам хоста: номера портов, имена файлов, учетные записи и т. д.;
- чтобы с помощью глобальных макросов обеспечить возможность изменения настроек в один щелчок.

Ниже приводится практический пример использования макроса уровня хоста в определении ключа элемента данных, такого как *Status of SSH daemon* (состояние демона SSH):

```
net.tcp.service[ssh,,{$SSH_PORT}]
```

Этот элемент автоматически будет связан со всеми хостами после определения макроса `{$SSH_PORT}` на уровне хоста. Таким способом осуществляется обобщение

элемента для любых хостов, на которых значение `$SSH_PORT` может отличаться; то же самое можно проделать в отношении HTTP-служб.

Импортирование и экспортирование шаблонов

Zabbix поддерживает весьма удобную и полезную возможность импорта/экспорта следующих объектов:

- **шаблонов**: включая все включенные в них элементы данных, триггеры, графики, комплексные экраны, правила обнаружения и присоединенные шаблоны;
- **хосты**: включая все включенные в них элементы данных, триггеры, графики, правила обнаружения и присоединенные шаблоны;
- **карты сетей**: включая все используемые изображения (экспорт/импорт карт поддерживается, начиная с версии Zabbix 1.8.2);
- **изображения**;
- **комплексные экраны**.

С помощью Zabbix API можно также экспортировать и импортировать группы хостов.

Функция экспорта проста и понятна, но функция импорта требует некоторых пояснений. Взгляните на рис. 7.6, на котором будет строиться дальнейшее обсуждение.

Раздел управления импортированием делится на три колонки: первая, **Update existing** (Обновлять существующие), обеспечивает обновление настроек уже имеющихся элементов. Эта функция особенно важна для желающих обновить элемент или просто добавить отсутствующие объекты. Вторая колонка, **Create new** (Создавать новые), как нетрудно догадаться, просто управляет добавлением новых элементов. Третья и последняя колонка, **Delete missing** (Удалять отсутствующие), появилась в версии Zabbix 2.4. Если выбран этот флажок, функция импортирования удалит все элементы, которые не были экспортированы, но присутствуют в текущей конфигурации.

Как видите, объекты шаблонов доступны для экспортирования/импортирования, и у нас есть возможность выбирать для экспорта **Template screens** (Экраны в шаблоне), **Template linkage** (Присоединенные шаблоны) и/или **Templates** (Шаблоны).

Присоединение шаблонов к хостам

Чтобы присоединить шаблон к хосту, нужно либо выбрать хосты в форме настройки шаблона, как было показано в разделе «Создание шаблонов», либо выбрать требуемый шаблон в форме настройки хоста, на вкладке **Template** (Шаблон).

После присоединения хост унаследует все сущности, присутствующие в шаблоне. *Сущности, прежде объявленные в конфигурации хоста, имена которых совпадают с именами сущностей в шаблоне, будут затерты*, но сущности, отсутствующие в шаблоне, останутся на месте и не будут затронуты операцией присоединения.

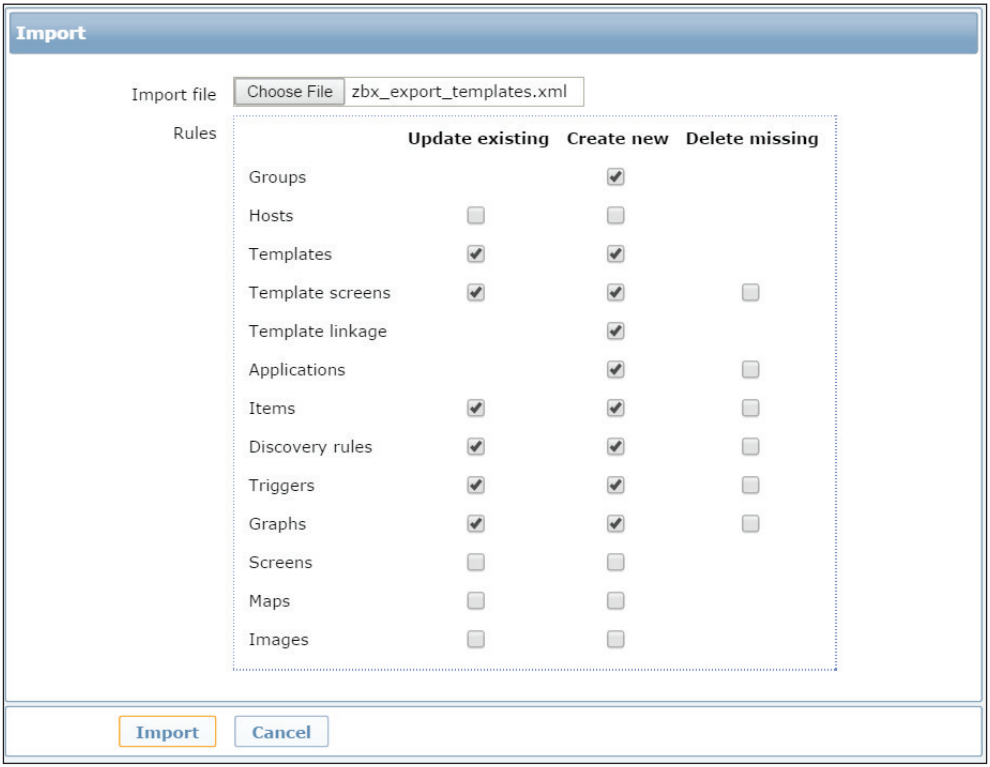


Рис. 7.6 ❖ Функция импортирования

Все сущности сохранят свои оригинальные имена, унаследованные из шаблона, при отображении в разделе веб-интерфейса с настройками, даже при просмотре формы с настройками хоста. Однако это не означает, что их изменение в форме настройки шаблона будет иметь тот же эффект, что изменение в форме настройки хоста.

Если сущность (элемент данных, триггер, график и т. д.) изменить в форме настройки шаблона, изменения немедленно отразятся на всех хостах, к которым присоединен данный шаблон. Но если сущность шаблона изменить в форме настройки конкретного хоста, изменения затронут только этот хост и не отразятся на конфигурации шаблона. Такой подход удобно использовать для тонкой настройки хоста, который во всех других отношениях является самым обычным, но при большом количестве мелких локальных изменений, которые трудно запомнить, может вызывать недопонимание. Кроме того, не все аспекты сущностей шаблонов доступны для изменения на уровне хоста. Можно изменить частоту обновления элемента данных, например, но нельзя изменить его ключ.

Процедура отсоединения шаблона по умолчанию не удаляет его сущности – для этого перед отсоединением необходимо выбрать флажок **Clear when unlinking** (Очистить при отсоединении). Будьте осторожны с этим флажком, так как опера-

ция с очисткой удалит все исторические данные и тренды. Если в ходе мониторинга были собраны какие-то фактические данные, лучше будет просто отсоединить шаблон от хоста и затем просто отключить все неиспользуемые элементы данных и триггеры, оставив исторические данные в неприкосновенности.

Вложенные шаблоны

Шаблоны можно присоединять не только к хостам, но и к другим шаблонам. Эта операция выполняется идентично присоединению шаблона к хосту; перейдите на вкладку **Linked templates** (Присоединенные шаблоны) в форме настройки шаблона и выберите шаблоны, которые желаете присоединить.

Эта операция может показаться ненужной, избыточной, но она очень удобна в двух случаях.

Во-первых, вложенные шаблоны позволяют еще больше обобщить пользовательские макросы. Поскольку шаблон наследует все сущности и свойства из присоединенных шаблонов, он также наследует пользовательские макросы и, соответственно, делает их доступными для хостов.

Рассмотрим конкретный пример. Допустим, что у нас имеется шаблон «Template Macros», содержащий, помимо всего прочего, пользовательский макрос `{$PFREE}` со значением `5`. Этот макрос можно было бы использовать для представления порогового значения процентов в проверке свободного дискового пространства, оперативной памяти или чего-то другого. Этот шаблон можно было бы присоединить к шаблонам «Template OS Linux» и «Template OS Windows» и использовать макрос `{$PFREE}` в их триггерах. С этого момента, если когда-нибудь потребуется изменить значение порога, достаточно просто изменить оригинальный шаблон «Template Macros», и обновленное значение автоматически распространится через присоединенные шаблоны до конкретных хостов.

Вторая, хоть и несколько ограниченная, область применения вложенных шаблонов – наследование не только макросов, но и всех остальных сущностей. Это может пригодиться при наличии общих наборов параметров для разных технологических уровней. Рассмотрим для примера случай, когда имеется большое количество практически идентичных физических серверов, действующих под управлением двух операционных систем (для простоты предположим, что это Linux и Windows), но выполняющих разные функции: серверы баз данных, файл-серверы, веб-серверы и т. д.

Можно, конечно, создать несколько монолитных шаблонов со всеми элементами данных, необходимыми для мониторинга серверов того или иного вида, включая проверки аппаратной части, операционной системы и приложений. Но можно пойти другим путем и создать своеобразную иерархию шаблонов. Общий шаблон с элементами для проверки аппаратной части, использующей механизм IPMI, будет наследоваться двумя шаблонами, специализированными для операционных систем. Эти два шаблона, в свою очередь, будут наследоваться шаблонами проверки приложений, например с такими именами, как «Linux Apache Template» или «Win Exchange Template». Эти шаблоны будут включать все элементы данных,

триггеры и графики, характерные для контролируемых ими приложений, в дополнение ко всем проверкам операционной системы, унаследованным из шаблона операционной системы, и аппаратным проверкам, унаследованным из общего шаблона аппаратного уровня. То есть, настраивая хост, вам все равно придется присоединить один шаблон, но у вас появится гибкая возможность создавать новые шаблоны и вкладывать их или изменять существующие только в одном месте и наблюдать результаты изменений по всей цепочке присоединенных шаблонов. Это также обеспечит наибольшую обобщенность, при сохранении возможности добавлять настройки, характерные для отдельных хостов.

Комбинирование шаблонов

Еще один способ поддержки модульного подхода к использованию шаблонов – создание конкретных шаблонов для всех технологических уровней и продуктов, но без связывания в иерархии на уровне шаблонов.

К любому хосту можно присоединить столько шаблонов, сколько потребуется, при условии что они не имеют конфликтов в именах или ключах элементов. По аналогии с предыдущим примером, хост «A» мог бы присоединять шаблон аппаратных проверок через IPMI, шаблон проверок для ОС Linux и шаблон проверок для сервера Apache, а хост «B» – шаблон аппаратных проверок через IPMI, шаблон проверок для ОС Linux и шаблон проверок для базы данных PostgreSQL.

Конечный результат получится тем же, что и при использовании вложенных шаблонов, о которых рассказывалось выше. Но какой подход предпочтительнее? Ответ на этот вопрос в значительной степени зависит от личных предпочтений, тем не менее при относительно небольшом количестве низкоуровневых шаблонов и достаточно высокой однородности аппаратных средств, операционных систем и технологических конфигураций решение на основе вложенных шаблонов оказывается проще в управлении. Вам понадобится только один раз присоединить шаблоны друг к другу и затем использовать полученные связки на большом количестве хостов. Такой подход можно также использовать с функцией обнаружения хостов, поскольку он упрощает присоединение шаблонов. С другой стороны, при наличии большого количества низкоуровневых шаблонов и богатом разнообразии технологических конфигураций возможность выбирать отдельные шаблоны для присоединения к хостам может оказаться предпочтительнее. Любые предварительно настроенные конфигурации могут оказаться слишком жесткими, чтобы быть полезными. Такой поход хорошо использовать, когда каждый хост приходится настраивать индивидуально из-за большого количества локальных особенностей и преимущества наследования выглядят весьма спорными.

Обнаружение хостов

Третий способ присоединения шаблонов к хостам – реализовать автоматическое присоединение, объединив механизм автоматического обнаружения хостов с действиями.

Механизм обнаружения в Zabbix состоит из набора правил, требующих периодически выполнять сканирование сети в поисках появления новых и исчезновения имевшихся хостов.

Zabbix поддерживает три метода проверки появления/исчезновения хостов в заданном диапазоне IP-адресов:

- проверка доступности агента Zabbix;
- проверка доступности агента SNMP;
- результаты простых внешних проверок (FTP, SSH и др.).

Эти проверки можно объединять, как показано на рис. 7.7.

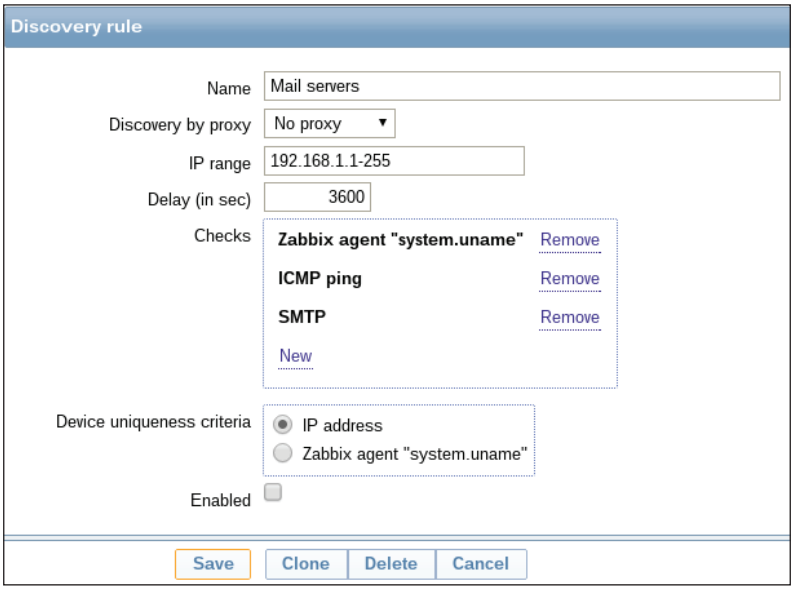


Рис. 7.7 ❖ Комбинирование разных видов проверок для обнаружения хостов

Как видите, это правило раз в час осуществляет проверку диапазона IP-адресов `192.168.1.1-255` на наличие серверов, которые:

- откликаются на запросы **ICMP ping**;
- имеют правильно настроенного агента Zabbix, возвращающего значение элемента `system.uname`;
- принимают запросы на порт SMTP, что характерно для серверов электронной почты.

Как это характерно для всех компонентов Zabbix, само правило обнаружения не выполняет никаких действий – оно лишь генерирует событие обнаружения. Это событие затем может перехватываться механизмом действий, которое должно решить, какие операции выполнить. Действия, обрабатывающие события обнаружения, практически идентичны действиям, обрабатывающим события триггеров.

Так как вы уже познакомились с действиями для обработки событий триггеров в *главе 6* «*Управление оповещениями*», далее мы коснемся только отличий, свойственных действиям для обработки событий обнаружения.

Первое отличие касается набора условий, чего и следовало ожидать (см. рис. 7.8).

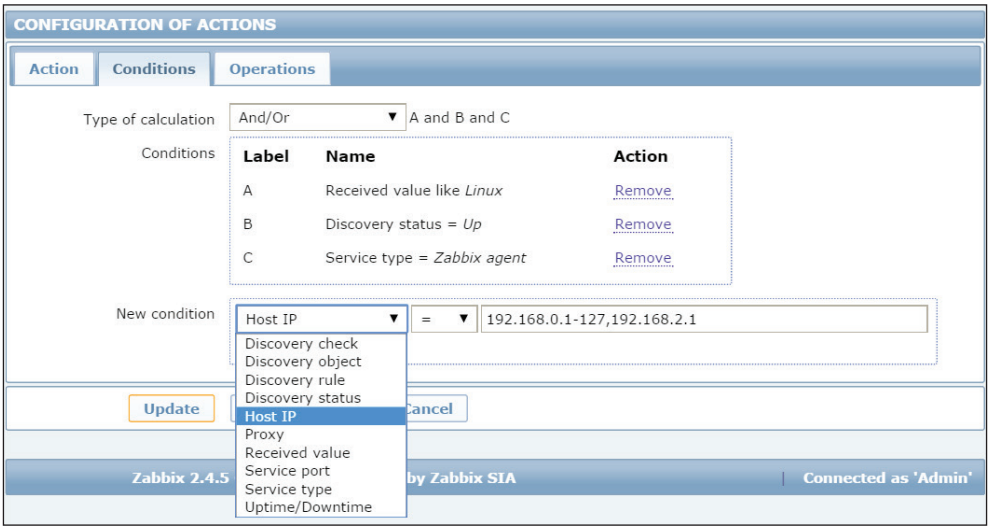


Рис. 7.8 ❖ Действия для обработки событий обнаружения имеют свой набор условий

Вместо имен хостов и параметров триггеров условия опираются на такие параметры, как: **Discovery status** (Состояние обнаружения), **Service type** (Тип сервиса) и **Uptime/Downtime** (Доступен/Недоступен). Условие **Received value** (Полученное значение) особенно интересно, так как позволяет, например, различать операционные системы, версии приложений и сопоставлять другие данные, получаемые от агента Zabbix или SNMP.

Эта информация становится особенно важной при настройке операций в действиях. Пример такой настройки показан на рис. 7.9.

Операции отправки сообщений и выполнения удаленных команд настраиваются в точности, как аналогичные операции в действиях, обрабатывающих события триггеров. С другой стороны, если с операциями добавления (или удаления) хоста все понятно, то когда дело доходит до добавления группы хостов или присоединения шаблона, становится очевидно, что богатый выбор действий, в зависимости от условия **Received value** (Полученное значение), может обеспечить инфраструктуре Zabbix весьма высокий уровень автоматизации.

Высокий уровень автоматизации наиболее полезен в быстро изменяющихся окружениях с хорошим уровнем предсказуемости в отношении типов хостов, таких как быстро растущие кластеры или решетки (grids). В таких окружениях каждый день могут появляться новые хосты и исчезать старые, но набор типов хостов

остается практически неизменным. Это идеальная предпосылка для создания небольшого набора правил обнаружения и действий, помогающих избежать необходимости постоянно вручную добавлять или удалять хосты одних и тех же типов. С другой стороны, если окружение достаточно стабильно или набор типов хостов постоянно изменяется, лучше все-таки больше внимания уделять конкретным хостам, так как ошибки в таких окружениях носят особенно критический характер.

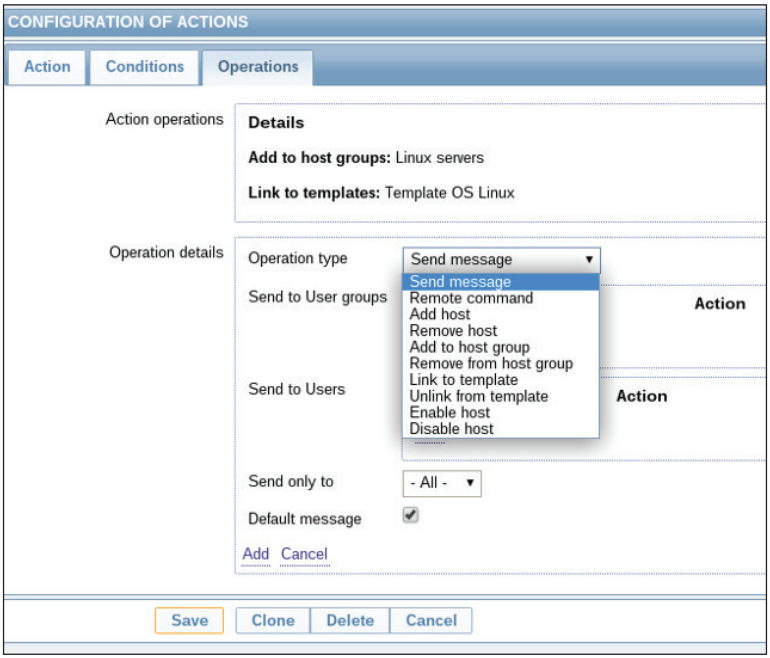


Рис. 7.9 ❖ Пример настройки операции

В подобных «хаотичных» окружениях или там, где вы не контролируете установку и развертывание новых систем, полезно ограничиться действиями, осуществляющими рассылку уведомлений об обнаружении хостов. Подобные уведомления о появлении или исчезновении хостов помогут обеспечить достаточно полный охват окружения мониторингом в случае недостаточно высокого уровня взаимодействия разных отделов предприятия.

Автоматическая регистрация активных агентов

В версии Zabbix 2.0 появилась возможность настраивать автоматическую регистрацию в агентах Zabbix, действующих в активном режиме. В этом случае новые хосты могут добавляться в систему мониторинга без необходимости настраивать их вручную на сервере. Автоматическая регистрация неизвестного хоста может выполняться в момент, когда агент запрашивает список проверок. Эта возможность может ока-

заться особенно ценной для реализации автоматического мониторинга новых узлов в облаке. Когда в облаке появляется новый узел, Zabbix автоматически начинает собирать данные о его производительности и проверять доступность хоста.

Поддержка автоматической регистрации с применением активных агентов может также применяться для мониторинга хостов с пассивными проверками. Для этого, запрашивая список проверок для выполнения, активный агент посылает серверу Zabbix свои параметры конфигурации `ListenIP` и `ListenPort`.

Если в конфигурационном файле агента указано несколько IP-адресов, на сервер будет отправлен только первый.

Сервер использует полученные IP-адрес и номер порта для настройки агента.

Если IP-адрес не будет получен сервером, он использует адрес, с которого установлено соединение. Если не будет получен номер порта, используется порт `10050`.

Настройка автоматической регистрации

Давайте посмотрим, как настроить автоматическую регистрацию. Сначала рассмотрим настройки на стороне агента. В конфигурационном файле агента нужно определить параметр `ServerActive`. Далее, если в `zabbix_agentd.conf` присутствует параметр `Hostname`, сервер будет использовать его значение для регистрации хоста, в противном случае будет использоваться системное имя узла сети.

На стороне сервера требуется настроить действие. Для этого перейдите на вкладку **Configuration ⇨ Actions** (Настройка ⇨ Действия), выберите источник события **Auto registration** (Авторегистрация) и щелкните на кнопке **Create action** (Создать действие). На рис. 7.10 показана форма с настройками после создания действия с именем «Active auto-registration».

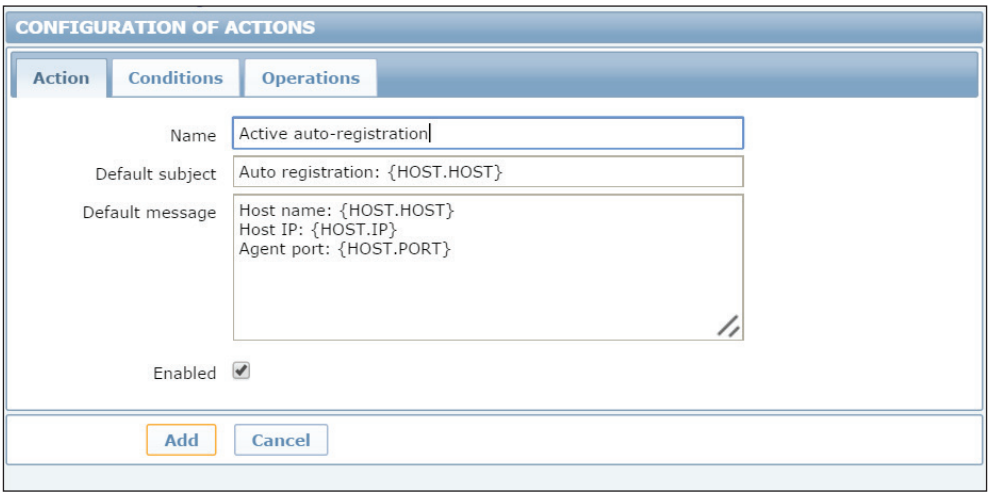


Рис. 7.10 ❖ Вновь созданное действие с именем «Active auto-registration»

Практический пример

Поле для экспериментов с автоматической регистрацией безгранично. Если хосты с автоматической регистрацией должны поддерживать только активный мониторинг (например, хосты, находящиеся за брандмауэром), для них имеет смысл определить специализированный шаблон, который будет автоматически присоединяться к ним. Давайте посмотрим, как это можно реализовать.

Для автоматизации настройки хоста можно определить параметры `HostMetadata` и `HostMetadataItem` на стороне агента. Представьте, что ко всем хостам с автоматической регистрацией, которые работают под управлением ОС Linux, требуется присоединить шаблон «Template OS Linux».

Для этого нужно добавить следующий параметр в файл `/etc/zabbix/zabbix_agentd.conf` на стороне агента:

```
HostMetadataItem=system.uname
```

Допустим также, что в нашем примере `HostMetadataItem` возвращает:

```
Linux servername.example.com 2.6.32-504.16.2.el6.x86_64 #1 SMP Wed Apr 22 06:48:29 UTC 2015
x86_64 x86_64 x86_64 GNU/Linux
```

В этом случае действие должно быть настроено в веб-интерфейсе, как показано на рис. 7.11.

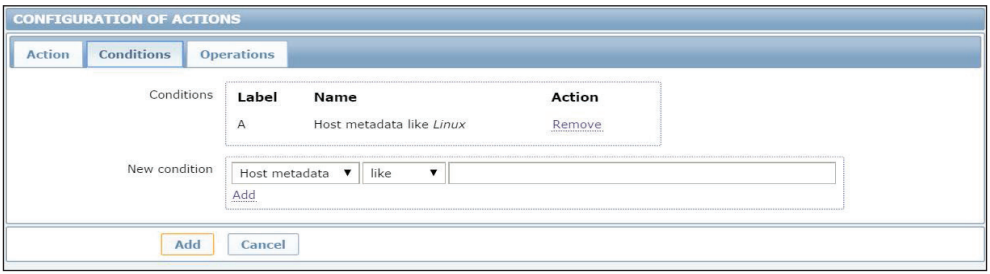


Рис. 7.11 ❖ Условие для действия автоматического обнаружения

С учетом условия **Host metadata like Linux** вкладка **Operations** (Операции) должна определять элементы, как показано на рис. 7.12.

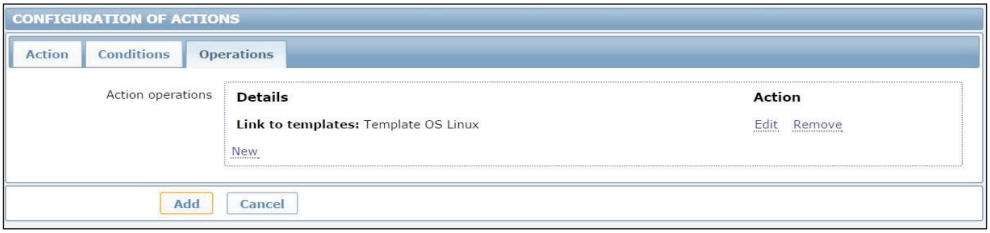


Рис. 7.12 ❖ Определение операции для действия автоматического обнаружения

Когда все условия на вкладке **Conditions** (Условия) будут удовлетворены, операция на вкладке **Operations** (Операции) присоединит шаблон «Template OS Linux» к хосту.

Если теперь включить в дистрибутив с агентом подготовленный конфигурационный файл, мы сможем значительно уменьшить время, затрачиваемое на настройку новых хостов.

Низкоуровневое обнаружение

Еще более важной и полезной особенностью шаблонов в Zabbix является поддержка специальных элементов, которые называются правилами низкоуровневого обнаружения. После определения этих правил они будут запрашивать у хостов списки настроенных ресурсов для мониторинга: файловые системы, сетевые интерфейсы, идентификаторы OID SNMP и др. Для каждого найденного ресурса сервер автоматически создаст элементы данных, триггеры и графики в соответствии со специальными прототипами, связанными с правилами.

Огромное преимущество правил низкоуровневого обнаружения заключается в том, что они принимают на себя все хлопоты, связанные с настройкой мониторинга наиболее изменчивых компонентов хостов, таких как типы и количество сетевых интерфейсов. Благодаря этому отпадает необходимость вручную создавать конкретные элементы данных и триггеры для каждого сетевого интерфейса или файловой системы хоста, или конструировать гигантские шаблоны со всевозможными элементами для данной операционной системы, которые в большинстве своем будут отключены. Вы сможете создать относительно небольшое количество обобщенных шаблонов, автоматически адаптирующихся под специфические особенности данного хоста, создавая «на лету» любые сущности для обнаруженных ресурсов.

По умолчанию Zabbix поддерживает четыре правила для обнаружения:

- сетевых интерфейсов;
- типов файловых систем;
- идентификаторов OID SNMP;
- процессоров и их ядер.

Так как правила обнаружения фактически являются особой разновидностью элементов, вы имеете возможность определять свои правила, учитывая их отличия от обычных элементов.

Помимо того что создание и настройка правил низкоуровневого обнаружения выполняются на вкладке **Discovery rules** (Правила обнаружения) в настройках шаблона, а не в обычном разделе **Items** (Элементы), эти два вида элементов отличаются еще и тем, что обычный элемент данных возвращает единственное значение, как рассказывалось в *главе 4 «Сбор данных»*, а элемент обнаружения всегда возвращает список пар макрос/значение в формате JSON. В этом списке перечисляются все ресурсы, найденные элементом обнаружения, вместе со ссылками на них.

В табл. 7.2 перечисляются элементы обнаружения, поддерживаемые в Zabbix по умолчанию, и возвращаемые ими значения (с некоторыми обобщениями), чтобы можно было понять, как создавать свои правила.

Таблица 7.2 ❖ Элементы обнаружения, поддерживаемые в Zabbix по умолчанию

Ключ элемента обнаружения	Тип элемента	Возвращаемое значение
vfs.fs.discovery	Агент Zabbix	{"data": [{"{#FSNAME}":"<path>", "{#FSTYPE}":"<fstype>"}, {"{#FSNAME}":"<path>", "{#FSTYPE}":"<fstype>"}, {"{#FSNAME}":"<path>", "{#FSTYPE}":"<fstype>"}, ...] }
net.if.discovery	Агент Zabbix	{"data":[{"{#IFNAME}":"<name>"}, {"{#IFNAME}":"<name>"}, {"{#IFNAME}":"<name>"}, ...]}
system.cpu.discovery	Агент Zabbix	{"data": [{"{#CPU.NUMBER}":"<idx>", "{#CPU.STATUS}":"<value>"}, {"{#CPU.NUMBER}":"<idx>", "{#CPU.STATUS}":"<value>"}, {"{#CPU.NUMBER}":"<idx>", "{#CPU.STATUS}":"<value>"}, ...] }
snmp.discovery	Агент SNMP (v1, v2 или v3)	{"data":[{"{#SNMPINDEX}":"<idx>", "{#SNMPVALUE}":"<value>"}, {"{#SNMPINDEX}":"<idx>", "{#SNMPVALUE}":"<value>"}, {"{#SNMPINDEX}":"<idx>", "{#SNMPVALUE}":"<value>"}, ...]}
custom.discovery	Любой	{"data":[{"{#CUSTOM1}":"<value>", "{#CUSTOM2}":"<value>"}, {"{#CUSTOM1}":"<value>", "{#CUSTOM2}":"<value>"}, {"{#CUSTOM1}":"<value>", "{#CUSTOM2}":"<value>"}, ...]}

Так же как для любых других элементов SNMP, ключ не имеет большого значения, главное требование – уникальность. Роль ключа в данном случае играет

значение SNMP OID; вы можете создавать разные правила обнаружения SNMP, которые выполняют поиск разных ресурсов, изменяя ключ элемента и отыскивая разные значения OID. Правило `custom.discovery` показано здесь в еще более абстрактном виде, поскольку оно зависит от фактического типа элемента.

Как видите, элемент обнаружения всегда возвращает список значений, но фактическое содержимое списка зависит от ресурса. В случае с файловой системой список будет содержать значения, такие как `{#FSNAME}:/usr,{#FSTYPE}:btrfs`, для каждой обнаруженной файловой системы. Правило обнаружения сетевых интерфейсов, с другой стороны, вернет список имен обнаруженных сетевых интерфейсов.

Настраивая правила обнаружения в шаблонах, нет необходимости беспокоиться ни о фактических значениях, возвращаемых в таких списках, ни о длине таких списков. Единственное, что вам следует знать, – это имена макросов, на которые можно ссылаться в прототипах. Прототипы – это вторая половина механизма низкоуровневого обнаружения. Они создаются точно так же, как обычные сущности шаблонов, и в них используются макросы механизма обнаружения, если это необходимо, как показано на рис. 7.13.

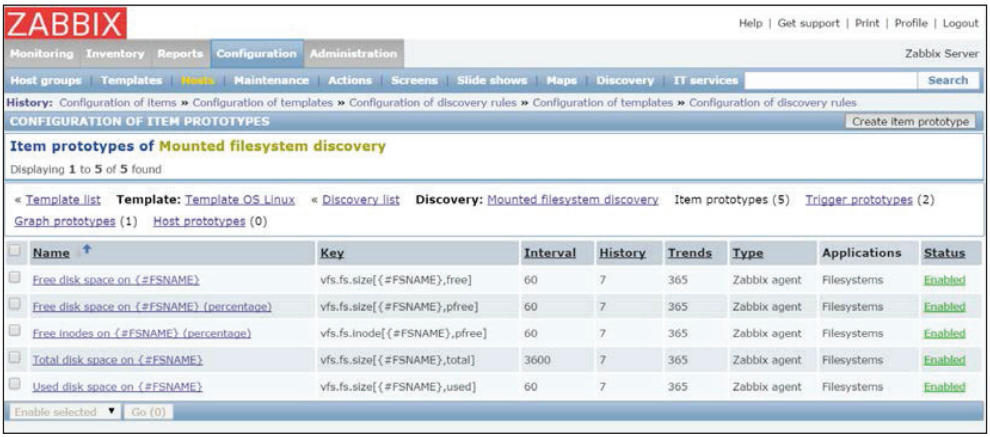


Рис. 7.13 ❖ Определение прототипов элементов

Когда шаблон применяется к хосту, на основе ресурсов, обнаруженных элементами обнаружения, создаются элементы данных, триггеры и графики, которые настраиваются в соответствии с прототипами.

Пользовательские правила обнаружения, с этой точки зрения, действуют точно так же, как пользовательские элементы данных: сценарии на стороне агента (и использующие нестандартный ключ `zabbix.agent`), внешние сценарии, запросы к базе данных и др. Единственное, что необходимо, – элементы должны возвращать значения в формате JSON, как показано в табл. 7.2, а в прототипах должны использовать ваши макросы.

Теперь посмотрим, как создать сценарий, реализующий простое низкоуровневое обнаружение.

В этом примере мы попытаемся с помощью механизма низкоуровневого обнаружения отыскать все жесткие диски, подключенные к физическому серверу. Прежде всего нам необходим сценарий, который должен быть установлен на стороне агента, который будет выводить результаты в формате JSON.

В данном случае это сценарий командной оболочки:

```
#!/bin/bash
disks=`ls -l /dev/sd* | awk '{print $NF}' | sed 's/[0-9]//g' | uniq`
elementn=`echo $disks| wc -w`
echo "{"
echo "\"data\":["
i=1
for disk in $disks
do
if [ $i == $elementn ]
then
echo "  {\"{#DISKNAME}\":\"$disk\",\"{#SHORTDISKNAME}\":\"${disk:5}\"}"
else
echo "  {\"{#DISKNAME}\":\"$disk\",\"{#SHORTDISKNAME}\":\"${disk:5}\"},"
fi
i=$((i+1))
done
echo "]"
echo "}"
```

Этот сценарий возвращает следующий результат:

```
{
"data":[
  {"{#DISKNAME}":"/dev/sda","{#SHORTDISKNAME}":"sda"},
  {"{#DISKNAME}":"/dev/sdb","{#SHORTDISKNAME}":"sdb"},
  {"{#DISKNAME}":"/dev/sdc","{#SHORTDISKNAME}":"sdc"},
  ...
]
}
```

Фактически сценарий перечисляет все имеющиеся устройства sd<X> и разделы.

Чтобы разрешить вызов сценария на стороне агента, нужно добавить в файл zabbix_agentd.conf следующие строки:

```
EnableRemoteCommands=1
UnsafeUserParameters=1
UserParameter=discovery.hard_disk,/<location-of-our-script>/discover_hdd.sh
```

Разумеется, после этих изменений агента Zabbix на удаленной машине следует перезапустить. А теперь определим правило обнаружения, как показано на рис. 7.14.

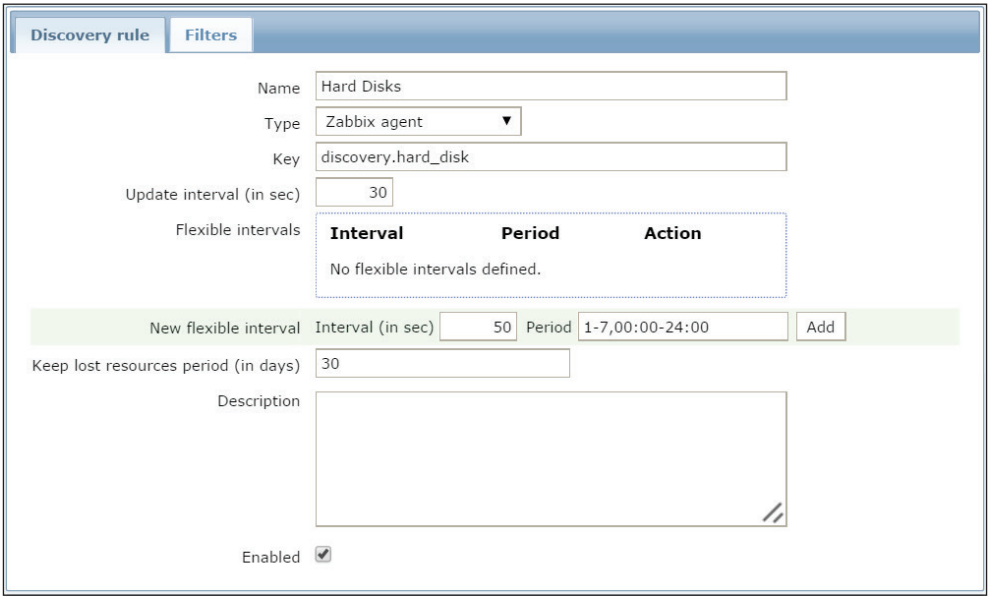


Рис. 7.14 ❖ Правило обнаружения жестких дисков

Затем, опираясь на найденные `#DISKNAME` или `#SHORTDISKNAME`, нужно определить прототипы элемента и триггера. Отличным примером для прототипирования может служить элемент, возвращающий количество операций ввода/вывода. Необходимые значения можно просто извлечь из файла `/proc/diskstats`:

```
$ grep sda /proc/diskstats |head -1 | awk '{print $12}'
19
```

Как видите, команда вернула количество операций ввода/вывода, выполняющихся в настоящий момент.

Дополнительную информацию о содержимом `/proc/diskstats` можно найти в официальной документации к ядру Linux, доступной по адресу: https://www.kernel.org/doc/Documentation/ABI/testing/procfs-diskstats.

Существует очень много интересных параметров, сбор и сохранение которых можно организовать для надежного управления инфраструктурой и планирования дальнейшего ее расширения. Для извлечения этих параметров можно зарегистрировать на стороне агента Zabbix переменную `UserParameter`. Ниже приводится одно из возможных множеств:

```
UserParameter=custom.vfs.dev.read.ops[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$4}'
UserParameter=custom.vfs.dev.read.ms[*],grep $1 /proc/diskstats | head -1 | awk '{print $$7}'
UserParameter=custom.vfs.dev.write.ops[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$8}'
```

```
UserParameter=custom.vfs.dev.write.ms[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$11}'
UserParameter=custom.vfs.dev.io.active[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$12}'
UserParameter=custom.vfs.dev.io.ms[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$13}'
UserParameter=custom.vfs.dev.read.sectors[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$6}'
UserParameter=custom.vfs.dev.write.sectors[*],grep $1 /proc/diskstats | head -1 | awk
'{print $$10}'
```

После этого можно перезапустить агента Zabbix и протестировать получение измеряемых параметров на стороне агента:

```
[root@ localhost ~]# zabbix_get -s 127.0.0.1 -k custom.vfs.dev.io.active[sda]
27
```

Теперь определим прототип элемента с использованием #SHORTDISKNAME, как показано на рис. 7.15.

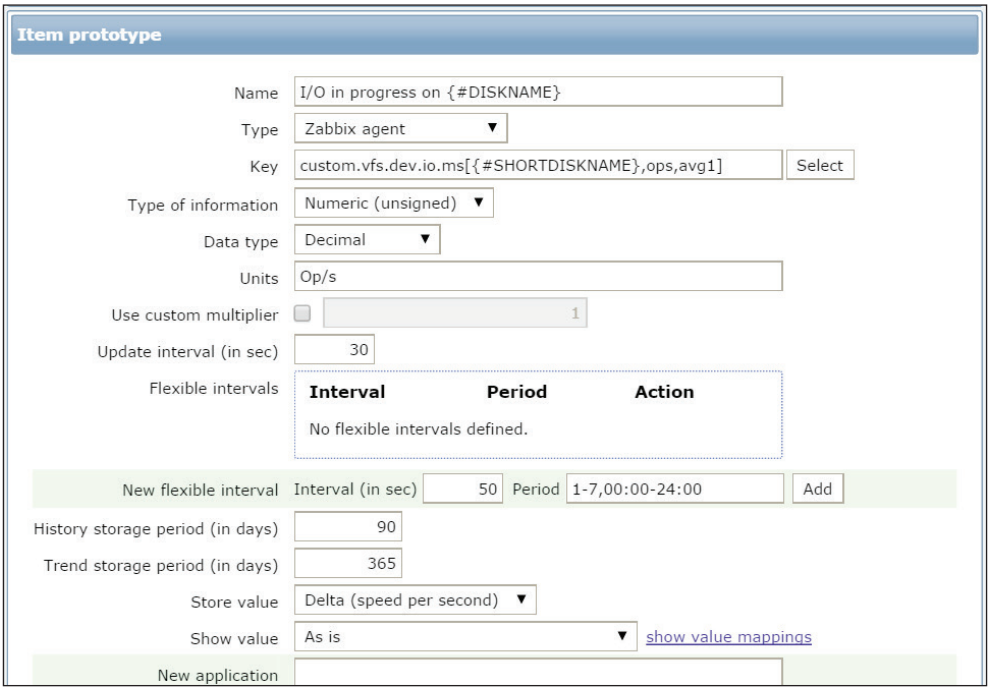


Рис. 7.15 ❖ Прототип элемента

Макрос {#SHORTDISKNAME} используется в определении ключа элемента, а в его имени используется макрос {#DISKNAME}. Обратите внимание, что переменная $1 в сценарии ссылается на его первый аргумент – ключ элемента. Аналогично мож-

но создать прототипы всех других зарегистрированных элементов. Настраивая правила обнаружения в шаблонах, нет необходимости беспокоиться ни о фактических значениях, возвращаемых в таких списках, ни о длине таких списков. Единственное, что вам следует знать, – это имена макросов, на которые можно ссылаться в прототипах.

Прототипы элементов, триггеров, графиков и хостов создаются точно так же, как обычные сущности шаблонов. Вам нужно только использовать макросы обнаружения, где это необходимо, а об остальном позаботится система Zabbix. Она создаст столько элементов данных, сколько их будет присутствовать в списке, возвращаемом правилом обнаружения для каждого прототипа элемента, столько триггеров, сколько их будет присутствовать в списке для каждого прототипа триггера, и т. д. (как показано на рис. 7.16).

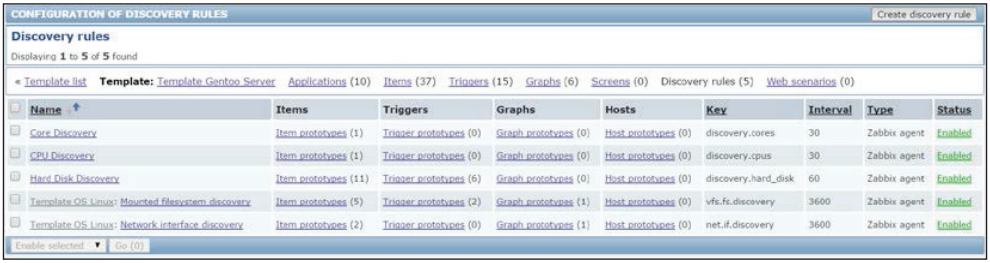


Рис. 7.16 ❖ Сущности, созданные на основе прототипов

С применением правил низкоуровневого обнаружения можно также создавать прототипы хостов, которые будут служить основой для реальных хостов после обнаружения серверов. Важно помнить, что до обнаружения реальных объектов прототипы не могут иметь собственных элементов данных и триггеров, помимо тех, что определены в присоединенных шаблонах. Обнаруженный хост превратится в реальный и унаследует IP-адрес существующего хоста.

В заключение

Эта глава завершает вторую часть книги, которая посвящена разработке и более подробному описанию основных функций системы мониторинга Zabbix. Эффективная настройка и использование шаблонов невозможны без знания особенностей элементов данных, графиков, комплексных экранов, триггеров и действий. К этим знаниям, полученным ранее, данная глава добавляет несколько дополнительных аспектов, имеющих отношение к шаблонам, знание которых поможет связать все предыдущие главы воедино. В настоящий момент вы получили все знания, необходимые для решения всех задач, связанных с реализацией и управлением системой мониторинга, – от выбора контролируемых параметров и настройки различных элементов данных, до определения элементов визуализации. Кроме того, вы узнали, как выбрать триггеры и действия, наиболее важные для

выразительности уведомлений, и как уменьшить вероятность ложных срабатываний. Наконец, у вас не должно возникать проблем с определением макросов и шаблонов для получения полноценной конфигурации, пригодной для мониторинга широкого спектра хостов, которая поддерживает автоматизацию операций посредством действий уровня хоста и механизма низкоуровневого обнаружения, действующего на уровне шаблона.

Заключительная часть книги посвящена обсуждению дополнительных возможностей настройки Zabbix, расширения функциональности системы и интеграции с другими системами управления для получения максимальной выгоды.

Следующая глава рассказывает о протоколе мониторинга и разработке сценариев расширения для Zabbix.

Глава 8

Внешние сценарии

К настоящему моменту вы познакомились с работой большинства серверных компонентов и узнали, как Zabbix извлекает данные из различных внешних источников. Теперь вы сможете настроить систему мониторинга в большом, разнородном и сложном окружении. Скорее всего, в вашей сети будут присутствовать разные нестандартные устройства и приборы. Обычно все эти устройства имеют некоторый интерфейс для взаимодействий с ними, но, к сожалению, часто случается так, что некоторые параметры нельзя извлечь из них с помощью **простого протокола сетевого управления (Simple Network Management Protocol, SNMP)** или другим стандартным способом.

Рассмотрим практический пример. В настоящее время все источники бесперебойного питания (ИБП) снабжены температурным датчиком, и есть вероятность, что в большом и разнородном окружении используются нестандартные ИБП, получить значение температуры с которых можно только с помощью инструмента, поставляемого производителем ИБП. Температура – критически важный параметр, особенно для больших ИБП. Поэтому очень важно следить за ее значением.

Представьте, что система охлаждения работает с перебоями; в этом случае возможность получить уведомление о превышении установленного температурного порога неоценима. Кроме того, предупреждение выхода оборудования из строя поможет сэкономить кучу денег. И даже если затраты на физическое восстановление оборудования не очень высоки, сам факт простоя может обойтись недешево и повлечь нежелательные потери для бизнеса. Отличным примером может служить торговая компания. В подобных компаниях оборудование должно работать безотказно, потому что в этой области идет жесткая борьба за производительность – способность выполнить покупку быстрее конкурентов на несколько миллисекунд является важным преимуществом. Нетрудно понять, что если серверы работают недостаточно хорошо, это уже проблема, а простой – настоящее бедствие для компании. Этот пример иллюстрирует, насколько важно уметь предупреждать отказы техники. Кроме того, важно понимать, какое большое значение имеет возможность извлекать все параметры функционирования инфраструктуры. Именно в таких ситуациях на помощь приходит Zabbix, предоставляя самые разные средства извлечения информации, взаимодействуя с операционной системой и позво-

ляя использовать инструменты командной строки. В частности, в арсенале Zabbix имеются следующие инструменты, помогающие удовлетворить описанные требования:

- внешние проверки (на стороне сервера);
- параметр настройки `UserParameter` (на стороне агента);
- `Zabbix_sender`: эту программу можно использовать и на стороне сервера, и на стороне агента;
- простой и эффективный протокол взаимодействий.

Эта глава целиком посвящена исследованию перечисленных инструментов и их применению для получения данных из внешних источников. Здесь вы узнаете, что не существует общего, оптимального и единственно верного решения всех проблем и каждое решение имеет свои положительные и отрицательные стороны. Эта глава познакомит вас со всем, что необходимо для реализации нестандартных проверок. Сведения, изложенные в этой главе, помогут вам выбрать решение, лучше всего подходящее для ваших условий.

В данной главе рассматриваются следующие темы:

- создание сценариев и их использование в качестве внешних сценариев;
- достоинства и недостатки сценариев, действующих на стороне сервера и на стороне агента;
- альтернативные способы отправки данных на сервер Zabbix;
- описание протокола Zabbix;
- учебная реализация протокола Zabbix с подробными комментариями.

Внешние проверки

Zabbix поддерживает средства извлечения данных, которые не могут быть получены с помощью стандартного агента. На практике нередко случается так, что нет никакой возможности установить стандартного агента на устройство, подлежащее мониторингу. Примерами таких устройств могут служить ИБП, все серверы, на которые по каким-то причинам нельзя устанавливать дополнительное программное обеспечение, или все приборы, вообще не поддерживающие возможности установки внешнего программного обеспечения.

Одним словом, если по какой-то причине вы не можете установить агента на устройство, но это устройство должно быть охвачено мониторингом, единственным доступным решением остается организация внешней проверки.

Местоположение сценария

Местоположение сценариев в системе Zabbix определяется настройками в конфигурационном файле `zabbix_server.conf`. По умолчанию, начиная с версии Zabbix 2.0, сценарии хранятся в каталоге `/usr/local/share/zabbix/externalscripts`.

Местоположение по умолчанию определяется с помощью переменной `datadir` как `${datadir}/zabbix/externalscripts`. Это правило действует и для сервера, и для прокси.

В предыдущих версиях использовался каталог по умолчанию /etc/zabbix/externalscripts. Но, как бы то ни было, его можно переопределить с помощью параметра ExternalScripts в файле конфигурации zabbix_server.conf:

```
### Параметр: ExternalScripts
# Обязательность: нет
# По умолчанию:
# ExternalScripts=${datadir}/zabbix/externalscripts
ExternalScripts=/usr/lib/zabbix/externalscripts
```

В версиях Zabbix 2.2 и 2.4 появилось еще несколько важных дополнений, касающихся внешних проверок:

- синтаксис ключа теперь поддерживает несколько параметров, перечисляемых через запятую;
- появилась поддержка пользовательских макросов в сценариях;
- пользовательские параметры, глобальные сценарии и внешние проверки теперь возвращают стандартную ошибку со стандартным выводом, которые можно анализировать внутри триггеров;
- появилась поддержка многострочных значений.

А теперь подробно рассмотрим, как действуют внешние проверки.

Особенности работы внешних проверок

Теперь самое время рассмотреть практический пример, чтобы понять, как действуют внешние проверки. Давайте просто попробуем определить количество файлов, открытых некоторым пользователем. Первое, что нужно сделать, – создать файл сценария в каталоге, указанном в параметре настройки ExternalScripts. Назовем сценарий lsof.sh и заполним его следующим кодом:

```
#!/bin/bash
if grep -q $1 /etc/passwd
    then lsof -u $1 | tail -n +2 |wc -l
else
    echo 0
fi
```

Эта программа принимает имя пользователя из первого аргумента командной строки, проверяет наличие такого пользователя в системе и затем возвращает количество файлов, открытых этим пользователем.

Теперь осталось только создать новый элемент данных с типом **External check** (Внешняя проверка). В поле **Key** (Ключ) введите lsof.sh["postgres"], как показано на рис. 8.1.

Перейдите на вкладку **Monitoring ⇨ Latest data** (Мониторинг ⇨ Последние данные) и понаблюдайте за данными, извлекаемыми сценарием (рис. 8.2).

Внешние сценарии должны возвращать ответ в разумные сроки, иначе элемент данных будет объявлен неподдерживаемым.

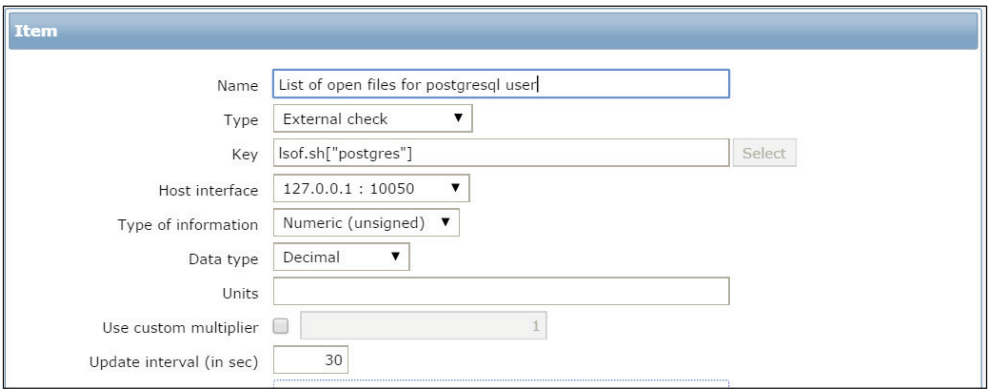


Рис. 8.1 ❖ Определение элемента данных для внешней проверки

Рис. 8.2 ❖ Данные, извлекаемые сценарием

До сих пор мы считали, что мониторинг с помощью внешнего сценария осуществляется непосредственно сервером Zabbix. Но имейте в виду, что если мониторинг хоста осуществляется прокси-сервером Zabbix, сценарий следует поместить на прокси-сервер, потому что в этом случае он должен выполняться там.

Теперь, когда вы знаете, как работают внешние сценарии, можно посмотреть, как с их помощью реализовать более сложные проверки.

В следующем примере мы настроим мониторинг удаленного экземпляра Oracle. Но для этого необходимо предварительно выполнить следующие настройки: установить клиента Oracle вместе с утилитами `sqlplus` и `tnsping`, а также создать учетную запись в целевой базе данных Oracle.

Последнюю версию обсуждаемого сценария можно получить на сайте http://www.smartmarmot.com/product/check_ora.

Я думаю, вам будет интересно узнать, как развился этот продукт, начиная с версии 1.0. Версия 1.0 доступна для загрузки на форуме Zabbix: https://www.zabbix.com/forum/showthread.php?t=13666.

Этот сценарий может служить отличным примером внешней проверки. Фактически, чтобы произвести все необходимые настройки, нужно выполнить следующие шаги:

1. Создать учетную запись во всех базах данных, подлежащих мониторингу.
2. Настроить клиента Oracle.
3. Распаковать и сохранить внешний сценарий в каталог, упоминавшийся выше.
4. Настроить параметры учетной записи в `<КАТАЛОГ_ДЛЯ_ВНЕШНИХ_СЦЕНАРИЕВ>/check_ora/credentials`.

5. Добавить определение хоста с именем, соответствующим имени экземпляра базы данных.

Последний пункт особенно важен и представляет особый режим использования Zabbix. Этот прием можно использовать всегда, когда требуется собрать параметры, которые принадлежат службе, а не фактическому хосту. Для большего реализма, если у вас есть СУБД, которая переключается на использование другого сервера в случае отказа, просто добавьте в конфигурацию Zabbix фиктивный хост с именем, совпадающим с именем базы данных. Если теперь произойдет отказ базы данных, процесс сбора данных продолжится, потому что процедура переключения выполняется автоматически. Этот метод применен здесь, потому что при правильной настройке клиент Oracle обрабатывает переключение автоматически.

Сделаем следующий шаг и добавим хост с именем, совпадающим с идентификатором SID. Например, если ваш экземпляр Oracle определен в файле `tnsnames.ora` как ORCL, добавьте хост с именем ORCL.

Zabbix позволяет создавать определения хостов с именами, соответствующими именам служб. Это дает возможность отделить службу от фактического хоста, который ее предоставляет.

Описание процедуры настройки клиента Oracle далеко выходит за рамки этой книги, поэтому она не будет обсуждаться здесь. Покончив с настройками, можно проверить сценарий, выполнив следующую команду:

```
check_ora.sh[-i <экземпляр> -q <запрос>]
```

Замените `<экземпляр>` именем экземпляра базы данных и `<запрос>` именем файла с запросом, который требуется выполнить. В каталоге `check_ora` можно найти много файлов с заранее подготовленными запросами; вы можете использовать их для мониторинга своей базы данных.

Использование Oracle SID или имени экземпляра Oracle в качестве имени хоста в Zabbix имеет свои преимущества. Соответствующее значение можно получить с помощью макроса `{HOSTNAME}`, благодаря чему в шаблоне легко можно определить ключ элемента, например: `check_ora.sh [-i {HOSTNAME} -q запрос]`.

Теперь, для извлечения результатов внешней проверки, добавьте в определение хоста элемент с ключом:

```
check_ora.sh[-i {HOSTNAME} -q <запрос>]
```

Например:

```
key="check_ora.sh[-i {HOSTNAME} -q lio_block_changes]"
```

Шаблон доступен на упоминавшемся форуме. Обратите внимание, что макрос `{HOSTNAME}` разворачивается в имя хоста, роль которого в данном случае играет имя экземпляра Oracle. С помощью макроса `{HOSTNAME}` можно создать обобщенный шаблон с элементами, которые будут распространяться на все хосты с базами данных.

Этот элемент данных будет действовать следующим образом:

1. Zabbix вызовет сценарий.
2. Сценарий выполнит следующие действия:
   - подключится к базе данных;
   - выполнит запрос и извлечет значение;
   - выведет значение на стандартный вывод;
3. Zabbix примет значение и, если оно допустимое, сохранит его.

Реализация сценария

Основой сценария `check_ora.sh` является функция `execquery()`:

```
execquery () {
start_time=$(date +%s)
# echo "Time duration: $((finish_time - start_time)) secs."
echo "BEGIN check_ora.sh $1 $2 `date`" >> /tmp/checkora.log
cd $SCRIPTDIR;
sqlplus -S $1 <<EOF | sed -e 's/^\ *//g'
  set echo off;
  set tab off;
  set pagesize 0;
  set feedback off;
  set trimout on;
  set heading off;
  ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '.,';
  @$2
EOF
finish_time=$(date +%s)
echo "END check_ora.sh $1 $2 `date`" >> /tmp/checkora.log
echo "check_ora.sh $1 $2 Time duration: $((finish_time - start_time)) secs." >> /tmp/
checkora.log
}
```

Она начинается с вывода информации в файл журнала `/tmp/checkora.log`:

```
start_time=$(date +%s)
# echo "Time duration: $((finish_time - start_time)) secs."
echo "BEGIN check_ora.sh $1 $2 `date`" >> /tmp/checkora.log
```

Она пригодится, когда потребуется выяснить, какая внешняя проверка и для какой базы данных выполняется. Также в файл журнала выводится время, потребовавшееся для выполнения операции:

```
finish_time=$(date +%s)
echo "END check_ora.sh $1 $2 `date`" >> /tmp/checkora.log
echo "check_ora.sh $1 $2 Time duration: $((finish_time - start_time)) secs." >> /tmp/
checkora.log
}
```

Поскольку файл является общим (для процессов check_ora.sh) и Zabbix может вызывать сценарии параллельно, важно явно отметить начало и конец операции, чтобы потом можно было точно установить начало и конец операции. Чтобы исключить разночтения, истекшее время вычисляется и выводится в последней строке.

Затем сценарий вызывает sqlplus:

```
sqlplus -S $1 <<EOF | sed -e 's/^\ *//g'
```

Здесь утилита sed отбрасывает начальные пробелы в выводе sqlplus. Это необходимо потому, что возвращаемое значение интерпретируется как число и потому не должно начинаться с пробелов; в противном случае элемент будет автоматически отключен!

Следующий фрагмент отключает вывод подробных сообщений клиентом Oracle:

```
set echo off;
set tab off;
set pagesize 0;
set feedback off;
set trimout on;
set heading off;
```

Это необходимо, чтобы не захламлять вывод ненужной информацией. Следующий фрагмент настраивает символы-разделители:

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = '.,';
```

Это необходимо потому, что базы данных могут настраиваться на использование разных кодировок символов. Кроме того, клиент может использовать другой символ-разделитель для дробных чисел. Поэтому нужно избежать неконтролируемых преобразований символов – это непреложное правило. Наконец, сценарий выполняет запрос из файла:

```
@$2
EOF
```

Результат отправляется в стандартный вывод и подбирается сервером Zabbix.

Основные правила создания сценариев

Этот сценарий охватывает все наиболее важные моменты, о которых следует помнить:

- избегайте попадания нежелательных символов в вывод;
- учитывайте типы значений, то есть, если ожидается число, удалите все ненужные символы (например, начальные пробелы);
- избегайте преобразования чисел в соответствии с региональными настройками, например точек и запятых;
- журналируйте проверки, помните, что внешние сценарии могут запускаться параллельно, из-за чего записи в журнале, принадлежащие разным проверкам, могут перемежаться между собой;
- фиксируйте в журнале время, потребовавшееся сценарию для выполнения;

- сценарии выполняются с привилегиями учетной записи сервера Zabbix, поэтому вам может понадобиться позаботиться о настройке разрешений для файлов сценариев и, возможно, подумать об использовании команды `sudo`.

Начиная с версии Zabbix 2.4 стандартный вывод сценариев объединяется со стандартным выводом ошибок; это может пригодиться для обработки ошибок и исключений, возникающих в сценариях.

Помните, что если требуемый сценарий отсутствует или сервер Zabbix не имеет прав для его выполнения, элемент данных будет объявлен неподдерживаемым. Кроме того, в случае превышения тайм-аута появится сообщение об ошибке, и процесс сценария будет автоматически остановлен.

Дополнительные замечания о внешних проверках

Выше в этом разделе вы видели, как выполняются внешние проверки и насколько сложные задачи, такие как мониторинг баз данных, они могут решать. Если вам требуется получить параметры, недостижимые другими способами, вы сможете получить их с помощью внешних проверок. Но, к сожалению, этот подход можно использовать не всегда. Кроме того, необходимо учитывать, что внешние проверки могут весьма интенсивно потреблять вычислительные ресурсы при широком их употреблении. Поскольку внешние проверки выполняются сервером Zabbix, они способны оказывать существенную нагрузку на него.

Чтобы запустить каждый внешний сценарий, сервер Zabbix выполняет ветвление (fork) процесса, поэтому выполнение большого количества сценариев может существенно ухудшить производительность сервера Zabbix.

Сервер Zabbix – основной компонент инфраструктуры мониторинга, поэтому не следует разбрасываться его ресурсами.

Параметр UserParameter

Чтобы избежать значительного потребления ресурсов сценариями, его можно перенести на сторону агента. Система Zabbix поддерживает альтернативный способ, основанный на применении параметра `UserParameter`, позволяющий перенести сценарий на сторону агента и разгрузить сервер Zabbix.

Параметр `UserParameter` определяется в конфигурационном файле агента. После его настройки он будет использоваться как любой другой агент Zabbix, посредством ключа, указанного в параметре. Ниже показано, как определяется параметр `UserParameter` в конфигурационном файле агента:

```
UserParameter=<ключ>,<команда>
```

Ключ в этом определении должен быть уникальным, а команда представляет выполняемую команду. Команда может быть составной и необязательно должна быть сценарием, например:

```
UserParameter=process.number, ps -e |wc -l
```

В данном примере определяется элемент с ключом process.number, который возвращает общее количество процессов, выполняющихся на сервере.

Используя тот же подход, можно вернуть количество подключенных пользователей:

```
UserParameter=process.number, who |wc -l
```

Гибкость параметра UserParameter

Используя такой метод, в конфигурационном файле агента можно определить много параметров UserParameter. Но это решение нельзя признать правильным, потому что конфигурационные файлы должны быть максимально простыми.

Чтобы избежать быстрого увеличения количества таких элементов, Zabbix реализует интересное свойство параметра UserParameter – дополнительную гибкость. Это свойство позволяет, например, определять такие значения:

```
UserParameter=ключ[*],<команда>
```

В данном случае ключ должен быть уникальным, а член [*] указывает, что данный ключ принимает параметры. Содержимое внутри квадратных скобок анализируется и подставляется в аргументы командной строки $1...$9 (учтите, что аргумент $0 хранит имя самой команды). Например, можно определить такой параметр UserParameter:

```
UserParameter=oraping[*],tnsping $1 | tail -n1
```

При обращении к этому элементу будет выполнена команда tnsping с идентификатором SID в аргументе $1. Тот же подход можно применить для определения количества пользователей:

```
UserParameter=process.number[*], ps -e |grep ^$1 | wc -l
```

Если понадобится переместить на сторону агента первый сценарий, возвращающий количество файлов, открытых указанным пользователем:

```
UserParameter=lsof.sh[*],/usr/local/bin/lsof.sh $1
```

Определив параметр UserParameter, не забудьте перезапустить агента, а на стороне сервера – изменить тип элемента и сохранить конфигурацию, как показано на рис. 8.3.

Аналогично можно настроить сценарий check_ora.sh для проверки базы данных:

```
UserParameter=check_ora.sk[*],check_ora.sh -i $1 -q $2
```

На стороне сервера Zabbix нужно создать элемент агента Zabbix (пассивного или активного) с ключом:

```
check_ora.sk[<имя_базы_данных> <запрос>]
```

Протестировать действие параметра UserParameter можно из командной строки, как описывалось прежде, или с помощью утилиты zabbix_get. Утилита zabbix_get выполняет проверку немедленно, избавляя от необходимости ждать обновления данных, и ее применение упрощает отладку происходящего на стороне агента.

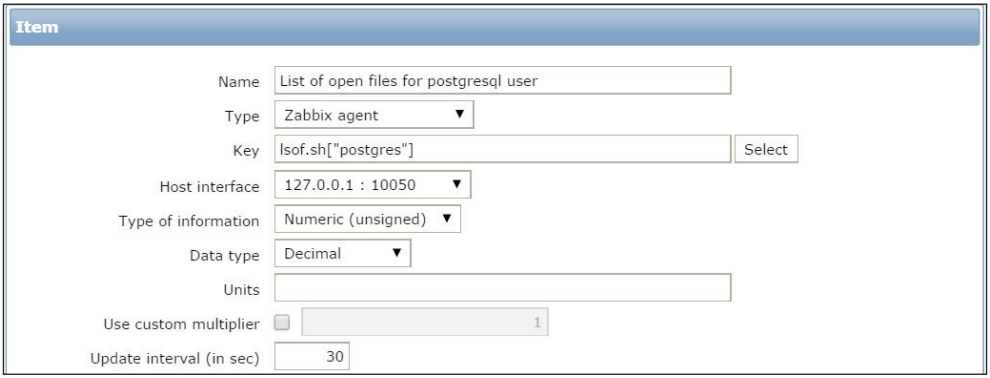


Рис. 8.3 ❖ Определение элемента в конфигурации сервера Zabbix

Проверить работу параметра `UserParameter` можно несколькими способами. Первый – с помощью утилиты `zabbix_get`; например, опробовать сценарий `losf.sh` с сервера Zabbix можно так:

```
# zabbix_get -s 127.0.0.1 -p 10050 -k lsof.sh["postgres"]
2116
```

Эта команда выведет результат операции. Также можно зарегистрироваться на стороне агента и выполнить следующую команду:

```
#/usr/sbin/zabbix_agentd -t lsof.sh["postgres"]
lsof.sh[postgres][/usr/local/bin/lsof.sh postgres] [t|2201]
```

Эта команда выведет вызванный сценарий и его вывод.

Замечания по использованию параметра UserParameter

Поддержка параметра `UserParameter` позволяет перенести сценарий со стороны сервера на сторону агента. Рабочая нагрузка, оказываемая сценарием, также переносится на сторону агента, благодаря чему экономятся ресурсы сервера. Таким способом можно распределить нагрузку между несколькими серверами. Очевидно, что при таком подходе все агенты будут контролировать базы данных, действующие на их хостах.

Параметр `UserParameter` действительно очень гибкий. Чтобы включить его на стороне агента, нужно изменить конфигурационный файл и перезапустить агента. Также необходимо гарантировать возврат значения требуемого типа, иначе оно будет отбрасываться.

Но, рассуждая о достоинствах, не нужно забывать о недостатках, например о так называемом эффекте наблюдателя (обсуждавшемся в *главе 1 «Развертывание Zabbix»*), возникающем при мониторинге таким способом. Команды мониторинга должны быть максимально легковесными, потому что агент выполняется на сервере, который оказывает основные услуги.

Использование параметра UserParameter подразумевает распределение сценариев между всеми серверами. В данном примере мониторинга Oracle нужно учитывать, сколько разных версий операционных систем и программ должно поддерживаться. Не исключено, что с течением времени потребуется поддерживать огромное количество разновидностей сценариев и программ. Это огромное количество вынудит вас централизовать развертывание, то есть хранить все версии сценариев в централизованном репозитории. Вам также придется позаботиться о нагрузке, добавляемой вашими сценариями, и если они не смогут обрабатывать все исключительные ситуации, справиться с этой задачей будет очень и очень сложно.

Параметр UserParameter – очень хороший и гибкий инструмент, и иногда действительно необходим для решения задач мониторинга, но он не предназначен для массового использования на каком-то одном хосте. Поэтому сейчас мы рассмотрим другой способ мониторинга элементов, которые не имеют встроенной поддержки в Zabbix.

Ниже приводятся важные замечания, касающиеся внешних сценариев и параметра UserParameter:

- все входные данные передаются сценарию в виде параметров и должны проверяться внутри него, чтобы предотвратить атаки вида «инъекция команд»;
- все полученные значения выводятся сценарием в стандартный вывод и должны быть отформатированы в соответствии с ожидаемыми типами; если сценарий ничего не вернет, это заставит сервер Zabbix объявить элемент неподдерживаемым;
- все сценарии должны выполняться максимально быстро;
- сценарии не должны совместно использовать ресурсы или блокировать их, а также производить другие побочные эффекты, чтобы избежать ситуаций «гонки за ресурсами» или ошибок из-за выполнения сразу нескольких экземпляров.

Отправка данных с помощью zabbix_sender

Итак, теперь вы знаете, как организовать внешние проверки на стороне сервера или агента. Но, как уже говорилось, ни один из этих методов не предназначен для массового использования, особенно если учесть, что в этой книге мы обсуждаем мониторинг больших окружений. Намного лучше было бы иметь сервер, специально выделенный для всех таких проверок.

В составе системы Zabbix имеется утилита, предназначенная для отправки данных на сервер, – zabbix_sender, – которая позволяет посылать элементы данных на сервер, где определены элементы ловушки (трапперы).

Чтобы протестировать утилиту zabbix_sender, просто добавьте элемент-ловушку в существующее определение хоста и выполните команду:

```
zabbix_sender -z <zabbixserver> -s <yourhostname> -k <item_key> -o <value>
```

В ответ вы получите сообщение примерно следующего содержания:

```
Info from server: "Processed 1 Failed 0 Total 1 Seconds spent 0.0433214" sent: 1; skipped:
0; total: 1
```

Как видите, пользоваться утилитой zabbix_sender очень просто. Теперь вы сможете выделить специализированный сервер для выполнения всех ресурсоемких сценариев.

Новый сценарий

Теперь можно изменить сценарий, использовавшийся прежде для внешней проверки, и добавить параметр UserParameter с новой версией, который обеспечит отправку данных в ловушку на сервере Zabbix.

Ниже приводится основная часть сценария:

```
CONNECTION=$( grep $HOST\; $CONNFILE | cut -d\; -f2) || exit 3;
RESULT=$( execquery $CONNECTION $QUERY.sql);
if [ -z "$RESULT" ]; then
    send $HOST $KEY "none"
    exit 0;
fi
send $HOST $QUERY "$RESULT"
exit 0;
```

Этот код выполняет следующие действия:

1) извлекает строку подключения из файла:

```
CONNECTION=$( grep $HOST\; $CONNFILE | cut -d\; -f2) || exit 3;
```

2) выполняет запрос, хранящийся в файле $QUER.sql:

```
RESULT=$( execquery $CONNECTION $QUERY.sql);
```

3) проверяет результат запроса и, если он не пустой, посылает его серверу Zabbix; иначе посылается значение "none":

```
if [ -z "$RESULT" ]; then
    send $HOST $KEY "none"
    exit 0;
fi
send $HOST $KEY "$RESULT"
```

В этом коде используются две основные функции: функция execquery(), которая практически не изменилась в сравнении с предыдущей версией, и функция send(). Функция send() играет ключевую роль в доставке данных на сервер Zabbix:

```
send () {
  MYHOST="$1"
  MYKEY="$2"
  MYMSG="$3"
  zabbix_sender -z $ZBX_SERVER -p $ZBX_PORT -s $MYHOST -k $MYKEY -o "$MYMSG";
}
```

Эта функция посылает значения с помощью утилиты командной строки zabbix_sender, в точности, как было показано в примере, когда мы проверяли ее. Посылаемому значению на сервере соответствует элемент ловушки, через который Zabbix примет данные и сохранит их.

Чтобы автоматизировать весь процесс проверки, необходимо написать сценарий-обертку, который будет выполнять опрос всех настроенных экземпляров Oracle, извлекать данные и посылать их серверу Zabbix. Сценарий-обертка получает список баз данных, с учетными данными для подключения к ним, из конфигурационного файла и в цикле вызывает сценарий check_ora_sendtrap.sh.

Сценарий-обертка для вызова check_ora_sendtrap

Так как этот сценарий предполагается запускать с помощью планировщика cron, в первую очередь необходимо настроить его окружение, загрузив конфигурационный файл:

```
source /etc/zabbix/externalscripts/check_ora/globalcfg
```

Затем перейти в каталог сценария. Обратите внимание, что из соображений совместимости структура каталогов не изменилась:

```
cd /etc/zabbix/externalscripts
```

После этого в цикле выполняются все запросы ко всем базам данных:

```
for host in $HOSTS; do
    for query in $QUERIES; do
        ./check_ora_sendtrap.sh -r -i $host -q ${query%.sql} &
        sleep 5
    done;
    ./check_ora_sendtrap.sh -r -i $host -t &
    sleep 5
    ./check_ora_sendtrap.sh -r -i $host -s &
done;
```

Обратите внимание, что этот сценарий выполняет все запросы, а также извлекает время tnsping и время соединения с каждой базой данных. Циклы выполняются по содержимому двух переменных окружения, хранящих списки хостов и запросов; они заполняются функциями:

```
HOSTS=$(gethosts)
QUERIES=$(getqueries)
```

Функция gethosts извлекает имена баз данных из конфигурационного файла /etc/zabbix/externalscripts/check_ora/credentials:

```
gethosts () {
  cd /etc/zabbix/externalscripts/check_ora
  cat credentials | grep -v '^#' | cut -d';' -f 1
}
```

Функция `getqueries` возвращает список файлов с запросами, присутствующих в каталоге:

```
getqueries () {
  cd /etc/zabbix/externalscripts/check_ora
  ls *.sql
}
```

Теперь осталось только запланировать вызов сценария-обертки, добавив в `crontab` следующую строку:

```
*/5 * * * * /etc/zabbix/externalscripts/check_ora_cron.sh
```

А ваш сервер Zabbix сохранит и отобразит данные.

Все сценарии, обсуждаемые здесь, доступны на условиях лицензии GPLv3 по адресам: https://sourceforge.net/projects/checkora и http://www.smartmarmot.com/.

Достоинства и недостатки выделенного сервера для внешних сценариев

Это решение основано на применении выделенного сервера для извлечения данных. То есть вы снимаете нагрузку с сервера, оказывающего услуги, а также с сервера Zabbix, и это очень хорошо.

К сожалению, этот подход не обладает большой гибкостью, и в данном конкретном случае все элементы данных обновляются раз в 5 минут. С другой стороны, частоту обновления данных с применением внешних проверок или параметра `UserParameter` можно настраивать для каждого элемента отдельно.

В данном конкретном случае, когда информация извлекается из баз данных, сценарий вносит известный эффект наблюдателя. Сам запрос может быть очень простым и легковесным, но, чтобы извлечь данные, `sqlplus` устанавливает соединение с Oracle. Это соединение используется всего несколько мгновений (необходимых для извлечения данных), после чего оно закрывается. Основная проблема заключается в отсутствии пулов соединений. Организовав кэширование соединений, вы сможете уменьшить влияние эффекта наблюдателя на базу данных.

Снижение нагрузки за счет буферизации соединений – широко известная идея, не зависящая от конкретной базы данных. Базы данных вообще демонстрируют снижение производительности, если их нагрузить большим количеством операций по установлению и закрытию соединений.

Идея организации пулов соединения достойна особого внимания. Чтобы лучше понять суть этой методологии, представьте сеть со сложной топологией, которая рассечена на сегменты множеством брандмауэров и маршрутизаторов, которые приходится преодолевать пакетам, несущим запрос на создание соединения. В таком случае становятся очевидными преимущества постоянных соединений. Наличие пула соединений, поддерживающего их открытыми с помощью вспомогательных пакетов, уменьшит задержки, возникающие при извлечении данных из

базы данных, и в целом снизит нагрузку на сеть. Процедура создания нового соединения включает проверку пакетов на всех пересекаемых брандмауэрах. Кроме того, имейте в виду, что если соединение устанавливается с Oracle, первый запрос на создание соединения посылается специальному процессу-слушателю (listener), который посылает клиенту ответный запрос и ждет от него подтверждения. К сожалению, кэширование соединений не может быть реализовано с применением инструментов командной оболочки. Существуют разные реализации таких пулов, но, прежде чем углубиться в программирование, нам нужно познакомиться с протоколами Zabbix.

Протоколы Zabbix

Протоколы Zabbix очень просты, и это очень важно, потому что простой протокол проще реализовать в своем собственном агенте или в программе, посылающей данные в Zabbix.

Zabbix поддерживает несколько версий протоколов. Их можно разделить на три семейства:

- Zabbix get;
- Zabbix sender;
- Zabbix agent.

Протокол Zabbix get

Протокол Zabbix get очень прост в реализации. Практически от вас требуется лишь посылать данные на сервер Zabbix, в порт `10050`.

Этот протокол настолько прост, что его можно реализовать даже в сценарии командной оболочки. Это текстовый протокол, использующийся для получения данных от агентов.

```
[root@zabbixserver]# telnet 127.0.0.1 10050
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
agent.version
ZBXD2.0.6
Connection closed by foreign host.
```

Данный пример демонстрирует, как узнать версию агента с помощью простой команды `telnet`. Обратите внимание, что ответ содержит заголовок `ZBXD`, за которым следуют фактические данные: `2.0.6`.

Этот простой протокол удобно использовать для получения данных непосредственно от агентов, с помощью сценариев командной оболочки.

Протокол позволяет узнать версию агента, не имея учетных данных для подключения к серверу, а также всех экземпляров `UserParameter`, которые определены на стороне агента.

Протокол Zabbix sender

Протокол Zabbix sender основан на передаче текста в формате JSON. Типичное сообщение состоит из следующих компонентов:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ><ДАННЫЕ>
```

Компонент <ЗАГОЛОВОК> имеет размер 5 байт и имеет вид ZBXD\x01. В действительности только первые четыре байта являются заголовком; пятый байт определяет версию протокола. В настоящее время поддерживается только версия 1 (0x01 в шестнадцатеричном виде).

Компонент <ОБЪЕМ_ДАННЫХ> занимает 8 байт и имеет шестнадцатеричный формат. Например, число 1 оформлено в виде 8-байтной последовательности 01/00/00/00/00/00/00/00.

Последний компонент – <ДАННЫЕ> – содержит текст в формате JSON.

Начиная с версии 2.0.3 Zabbix может принимать данные, объем которых не превышает 128 МБ. Цель этого ограничения – предотвратить исчерпание оперативной памяти на сервере из-за получения огромных объемов данных.

Чтобы послать значение, требуется сконструировать JSON-сообщение следующего вида:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "request":"sender data",
  "data":[
  {
    "host":"Host name 1",
    "key":"item_key",
    "value":"XXX",
    "clock":unix_time_format
  },
  {
    "host":"Host name 2",
    "key":"item_key",
    "value":"YYY"
  }
  ],
  "clock":unix_time_format
}
```

Как видите, в одно сообщение можно упаковать несколько значений, даже если они получены с разных хостов или имеют разные ключи.

Член "clock" является необязательным, и его можно опустить.

После приема данных сервер должен отправить ответ, имеющий следующую структуру:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "response":"success",
  "info":"Processed 6 Failed 1 Total 7 Seconds spent 0.000283"
}
```

Ниже приводится несколько замечаний относительно ответа:

- ответ включает статус `[success|failure]`, который относится к передаче всего списка элементов;
- иногда, как показано в примере, прием некоторых элементов данных может потерпеть неудачу; в этом случае самое большее, что можно сделать, – это зарегистрировать ответ в файле журнала.

Важно следить за временем, потраченным на отправку списка элементов, потому что увеличение этого интервала может говорить о высокой загруженности сервера Zabbix.

К сожалению, этот протокол не сообщает ничего о том, какие элементы данных не были приняты и по каким причинам. На момент написания этих строк все еще оставались неудовлетворенными два запроса на совершенствование:

- реализовать более читаемый вывод: https://support.zabbix.com/browse/ZBXNEXT-935;
- идентифицировать непринятые элементы: https://support.zabbix.com/browse/ZBXNEXT-246.

Теперь вы знаете, как действует протокол Zabbix sender, в версии Zabbix 1.8 и выше.

Еще одна проблема состоит в том, что протокол Zabbix sender до сих пор не поддерживает шифрования, из-за чего могут возникать негативные последствия при передаче конфиденциальных данных в открытом виде. Необходимо также учитывать вероятность подделки сообщений хакерами с целью скрыть свою деятельность за большим количеством тревог и уведомлений. Используя этот протокол, злоумышленник легко сможет посылать ложные тревоги, чтобы заставить срабатывать триггеры, и за этим фоном скрыть свои действия.

К счастью, этот недостаток ликвидируется в настоящее время, и команда разработчиков работает над реализацией версий протокола с поддержкой SSL и TLS.

За дополнительной информацией обращайтесь к запросу: https://support.zabbix.com/browse/ZBXNEXT-1263.

Интересная недокументированная особенность

Существует одна очень интересная недокументированная и малоизвестная особенность протокола Zabbix sender. Углубляясь в изучение протокола, в первую очередь следует ознакомиться с официальной документацией, а во вторую – исследовать его реализацию. Может так получиться, что не все последние изменения нашли отражение в документации.

Итак, рассмотрим код `zabbix_sender`. В нем можно найти раздел с реализацией протокола:

```
zbx_json_addobject(&sentdval_args.json, NULL);
zbx_json_addstring(&sentdval_args.json, ZBX_PROTO_TAG_HOST,
                   hostname, ZBX_JSON_TYPE_STRING);
zbx_json_addstring(&sentdval_args.json, ZBX_PROTO_TAG_KEY, key,
                   ZBX_JSON_TYPE_STRING);
zbx_json_addstring(&sentdval_args.json, ZBX_PROTO_TAG_VALUE,
                   key_value, ZBX_JSON_TYPE_STRING);
```

Предыдущий фрагмент реализует формирование сообщения протокола Zabbix sender в формате JSON, точнее следующий его раздел:

```
"host":"Host name 1",
"key":"item_key",
"value":"XXX",
```

До этой точки протокол хорошо документирован. Но сразу за этими строками следует любопытный раздел, реализующий еще одну особенность элемента JSON.

```
if (1 == WITH_TIMESTAMPS)
  zbx_json_adduint64(&sentdval_args.json, ZBX_PROTO_TAG_CLOCK, atoi(clock));
```

Этот код извлекает отметку времени из элемента и добавляет ее в виде свойства объекта JSON, после чего сообщение закрывается:

```
zbx_json_close(&sentdval_args.json);
```

Отметка времени определена как значение типа `int64`.

Это очень важная особенность, поэтому, если вы соберетесь написать собственную реализацию `zabbix_sender`, вы должны будете добавить отметку времени, соответствующую моменту приема элемента данных.

Обратите внимание, когда будете тестировать этот фрагмент, что Zabbix сохраняет время извлечения элемента, полученное от базы данных.

Свойство clock в объектах JSON

Теперь это свойство можно использовать для оптимизации. Сервер Zabbix может в одной посылке принять до 128 МБ данных. Конечно, лучше не приближаться к этому пределу, потому что его достижение является признаком плохой реализации.

Особенность, связанная с поддержкой свойства `clock` и описанная выше, может использоваться в двух сценариях:

- если нужно послать элементы в одном пакете;
- если сервер недоступен, данные можно кэшировать и послать их позже.

В первом случае это свойство помогает оптимизировать передачу данных и уменьшить количество операций подключения к серверу.

Во втором случае обеспечиваются надежность и отказоустойчивость отправителя, который корректно обрабатывает случаи выхода из строя сервера Zabbix

и сохраняет неотправленные элементы в кэше, которые будут отправлены с восстановлением работоспособности сервера. Но помните, что после долгого простоя сервера отправитель не должен затопить сервер кэшированными данными. Посылайте в каждом пакете разумное количество элементов.

Протокол Zabbix agent

Этот протокол немного сложнее, потому что предусматривает большее количество разновидностей диалогов и этапов в них. Когда запускается активный агент, в первую очередь он должен подключиться к серверу и запросить задание для выполнения, точнее, список извлекаемых элементов и периоды времени.

Протокол Zabbix agent имеет такой же формат, как и протокол Zabbix sender:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ><ДАННЫЕ>
```

Компоненты <ЗАГОЛОВОК>, <ОБЪЕМ_ДАННЫХ> и <ДАННЫЕ> имеют то же назначение, как рассказывалось в предыдущем разделе.

Диалог начинается с отправки агентом запроса:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "request":"active checks",
  "host":"<hostname>"
}
```

Таким способом агент запрашивает имя хоста, соответствующее активному списку элементов для проверки. Сервер может вернуть ответ, например, следующего содержания:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "response":"success",
  "data":[{
    "key":"log[\/var\/log\/localmessages,@errors]",
    "delay":1,
    "lastlogsize":12189,
    "mtime":0
  },
  {
    "key":"agent.version",
    "delay":"900"
  }]
  "regexp":[
  {
    "name":"errors",
    "expression":"error",
    "expression_type":0,
    "exp_delimiter":",",
    "case_sensitive":1
  }]
}
```

Сервер Zabbix должен вернуть статус success и список с ключами элементов и задержками.

Для элементов типа log и logrt сервер должен указать значение lastlogsize. Этот параметр необходим агенту для работы. Кроме того, для всех элементов logrt должно быть указано также значение mtime.

Компонент "regexp", присутствующий в данном примере, возвращается агенту, только если вы определили глобальные или регулярные выражения. Обратите внимание, что если в определении ключа используется пользовательский макрос, он разворачивается на сервере, а исходный ключ (с макросом в определении) посылается агенту в свойстве key_orig.

Получив ответ от сервера, агент закрывает TCP-соединение, анализирует сообщение и приступает к сбору данных с указанной периодичностью. После сбора данных элементы отсылаются обратно на сервер:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "request":"agent data",
  "data":[
  {
    "host":"HOSTNAME",
    "key":"log[\/var\/log\/localmessages]",
    "value":"Sep 16 18:26:44 linux-h5fr dhcpcd[3732]: eth0: adding default
    route via 192.168.1.1 metric 0",
    "lastlogsize":4315,
    "clock":1360314499,
    "ns":699351525
  },
  {
    "host":"<hostname>",
    "key":"agent.version",
    "value":"2.0.1",
    "clock":1252926015
  }
  ],
  "clock":1252926016
}
```

Реализуя этот протокол, не забывайте посылать назад lastlogsize для всех элементов типа log и mtime – для элементов logrt.

Сервер ответит сообщением:

```
{
  "response":"success",
  "info":"Processed 2 Failed 0 Total 2 Seconds spent 0.000110"
}
```

Сервер может сообщить, что какие-то элементы не были приняты, но в настоящий момент невозможно узнать – какие.

Еще некоторые варианты ответов

Для полноты описания протокола ниже приводится еще несколько вариантов ответов, о которых следует знать:

- когда хост не обслуживается системой мониторинга;
- когда определение хоста отсутствует;
- когда для хоста настроен активный режим мониторинга, но отсутствуют определения активных элементов.

В первом случае, когда хост не обслуживается системой мониторинга, агент получит следующий ответ от сервера:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "response":"failed",
  "info":"host [Host name] not monitored"
}
```

Во втором случае, когда определение хоста отсутствует, агент получит следующий ответ:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "response":"failed",
  "info":"host [Host name] not found"
}
```

И в последнем случае, когда для хоста настроен активный режим мониторинга, но отсутствуют определения активных элементов, агент получит пустой набор данных:

```
<ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ>{
  "response":"success",
  "data":[]
}
```

Протокол низкоуровневого обнаружения

Протокол низкоуровневого обнаружения обеспечивает поддержку автоматического создания элементов, триггеров и графиков для разных сущностей. Например, Zabbix может автоматически приступить к мониторингу файловых систем или сетевых интерфейсов на вашей машине, не требуя предварительного создания вручную элементов для всех файловых систем или сетевых интерфейсов. Фактически в результате работы функции обнаружения может быть выполнено множество разных действий, таких как удаление ненужных сущностей, например элементов и т. д. Эти возможности придают немалую гибкость системе мониторинга. Zabbix действительно позволяет создавать и настраивать совершенно новые правила низкоуровневого обнаружения сущностей Zabbix любого типа.

Рассмотрим вывод, полученный в результате работы элемента низкоуровневого обнаружения, такого как `vfs.fs.discovery`. Чтобы получить этот вывод, достаточно выполнить следующую команду:

```
$ zabbix_get -s 127.0.0.1 -k vfs.fs.discovery
{"data":[
  {"{#FSNAME}":"/","{#FSTYPE}":"rootfs"},
  {"{#FSNAME}":"/proc","{#FSTYPE}":"proc"},
  {"{#FSNAME}":"/sys","{#FSTYPE}":"sysfs"}
  ...
]}]
```

Я немного сократил вывод, но в любом случае, как можно видеть, он легко читается. Во-первых, это текст в формате JSON, содержащий данные в виде ключ/значение.

Как было показано в *главе 7* «*Управление шаблонами*», мы можем создать любые сценарии, необходимые для обнаружения сущностей, подлежащих мониторингу.

Разумеется, каждый сценарий, действующий на стороне агента, должен быть зарегистрирован как `UserParameter` в файле `zabbix_agent.conf`. Если сценарии действуют на стороне сервера, они должны храниться в каталоге, определяемом параметром `ExternalScripts` в файле `zabbix_server.conf`.

Рассмотрим еще один пример сценария низкоуровневого обнаружения, который можно использовать для выявления всех открытых портов в вашей сети. Как рассказывалось в *главе 7* «*Управление шаблонами*», нам нужно сформировать сообщение в формате JSON, перечисляющем открытые порты и названия соответствующих протоколов. Получить эту информацию можно с помощью `nmap`. Установка `nmap` в Red Hat выполняется следующей командой от имени root:

```
$ yum install nmap
```

Эта команда установит только внешний компонент. Сценарий поиска всех открытых портов лучше запускать на сервере Zabbix, так как некоторые порты на хостах могут быть открыты локально и закрыты для внешних наблюдателей с помощью брандмауэра. Команда быстрого сканирования портов использует параметр `-T<0-5>`, определяющий шаблон хронометража (чем больше значение, тем более быстрый шаблон используется). В данном сценарии используется параметр `-T4`, а также параметр `-F` (fast mode – быстрый режим):

```
#!/bin/sh
# Начало заголовка JSON
echo '{'
echo ' "data":['
PORTS=( $(nmap -T4 -F ${1} | grep 'open' | cut -d" " -f1 ) )
COUNTER=${#PORTS[@]}
for PORT in "${PORTS[@]}"; do
    COUNTER=$(( COUNTER - 1))
    if [ $COUNTER -ne 0 ]; then
        echo ' { "{#PORT}":"'$(echo $PORT| cut -d/ -f1)'", "{#PROTO}":"'$(echo $PORT| cut
-d/ -f2)'" },'
    else
# это последний элемент,
# который, согласно требованиям JSON, не должен содержать
```

```
# завершающую запятую
        echo ' { "{#PORT}":"'$(echo $PORT| cut -d/ -f1)'", "{#PROTO}":"'$(echo $PORT| cut
-d/ -f2)'" }'
    fi
done
# Конец сообщения JSON
echo ' ]'
echo '}'
```

Сценарий сканирует порты хоста с указанным IP-адресом и возвращает номера тех, что не закрыты брандмауэром, а также соответствующие им имена протоколов:

```
# ports_ldd.sh 192.168.1.1
{
  "data":[
    { "{#PORT}":"22}", "{#PROTO}":"tcp" },
    { "{#PORT}":"25}", "{#PROTO}":"tcp" },
    { "{#PORT}":"53}", "{#PROTO}":"tcp" },
    { "{#PORT}":"80}", "{#PROTO}":"tcp" },
    { "{#PORT}":"111}", "{#PROTO}":"tcp" },
    { "{#PORT}":"631}", "{#PROTO}":"tcp" },
    { "{#PORT}":"3306}", "{#PROTO}":"tcp" },
    { "{#PORT}":"5432}", "{#PROTO}":"tcp" }
  ]
}
```

Именно такой вывод мы ожидали получить. Разумеется, сценарий должен находиться в каталоге, указанном в параметре настройки `ExternalScripts`. Теперь можно создать правило обнаружения на вкладке **Discovery rule** (Правила обнаружения), как показано на рис. 8.4.

Внутри прототипов уже определены две переменные: `{#PORT}` и `{#PROTO}`, а теперь создадим прототип элемента данных со следующими свойствами:

- **Name** (Имя): `Status of port {#PORT}/{#PROTO}`;
- **Type** (Тип): **Simple check** (Простая проверка);
- **Key** (Ключ): `net.tcp.service[{#PROTO},,{#PORT}]`;
- **Type of information** (Тип информации): **Numeric (unsigned)** (Числовой (целое положительное));
- **Data type** (Тип данных): **Boolean** (Логический).

Определение прототипа элемента показано на рис. 8.5.

Также нужно определить прототип триггера:

- **Name** (Имя): `{#PROTO} port {#PORT}`;
- **Expression** (Выражение): `{Template_network:net.tcp.service[{#PROTO},,{#PORT}].last(0)}=0`.

После выполнения описанных настроек функция обнаружения отыщет все открытые порты, достижимые для сервера, и создаст элементы данных с триггерами, которые будут срабатывать при недоступности портов.

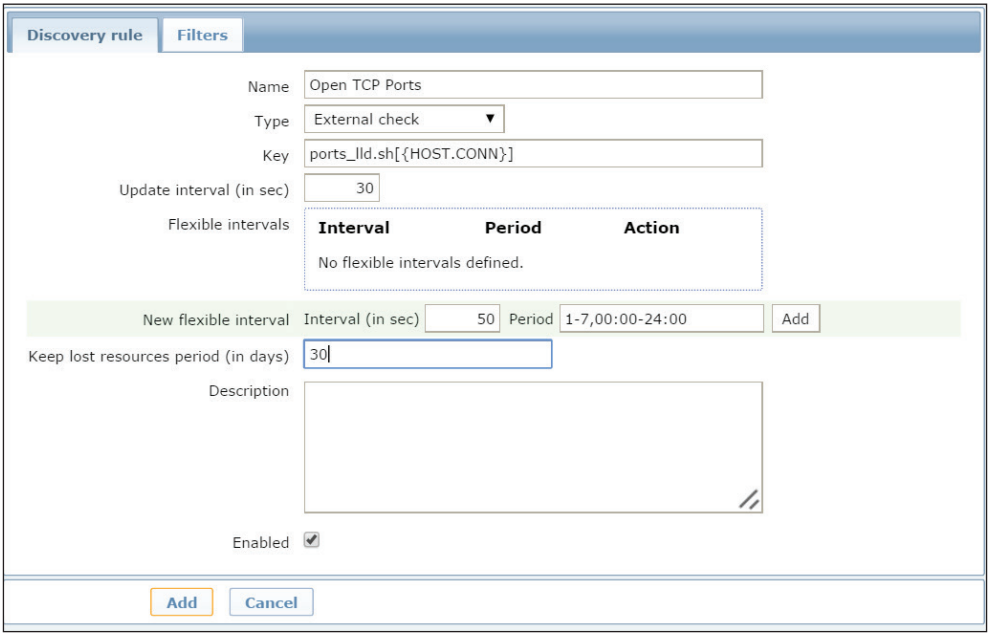


Рис. 8.4 ❖ Правило обнаружения открытых портов

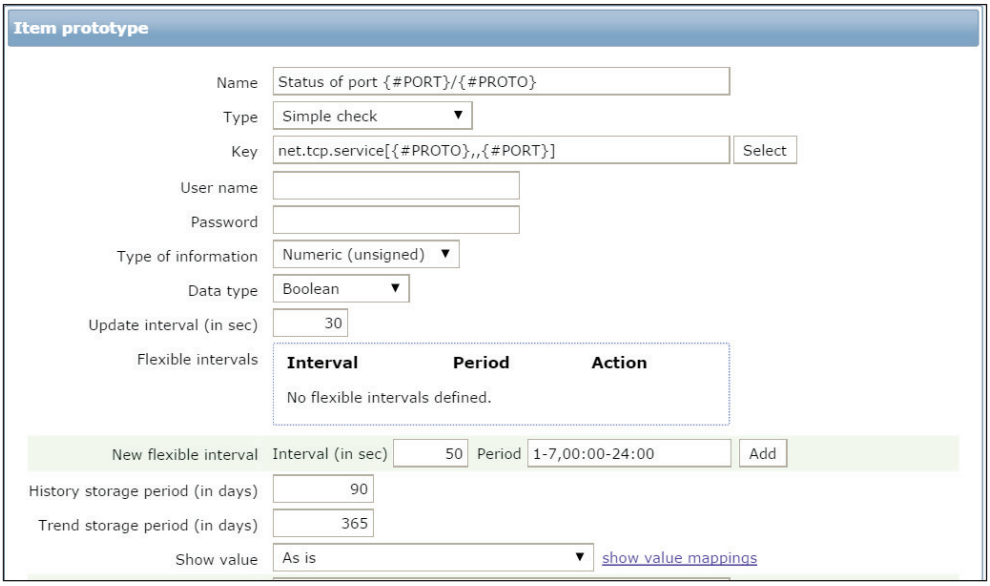


Рис. 8.5 ❖ Определение прототипа элемента

Здесь мы сымитировали ситуацию, когда требуется организовать мониторинг всех служб, доступных серверу, и посылать уведомление, если порт становится недоступным. Важно рассмотреть также другой случай, когда службы размещаются в защищенном сегменте сети (DMZ) и требуется посылать уведомление, если по какой-то причине служба стала доступна извне этого сегмента. Типичным примером может служить порт подключения к базе данных, который должен быть доступен только изнутри защищенного сегмента сети, но не извне.

Это был пример очень простой автоматизации, но его можно расширить. Представьте сеть, где присутствует четко определенный сегмент служб, и вам известно, какие демоны используются, или, по крайней мере, вы точно знаете, что баннеры демонов не были изменены для сокрытия назначения служб. В этом случае можно было бы реализовать поиск всех открытых портов и после идентификации служб, использующих эти порты, применять соответствующий шаблон для мониторинга. Например, если обнаружится открытый порт 80, используемый службой Apache, к этому хосту можно применить шаблон «Apache». Подобное решение обеспечило бы еще более широкую автоматизацию и сократило средства и время, необходимые для запуска системы мониторинга.

Взаимодействие с Zabbix

Теперь, зная особенности работы протокола Zabbix, займемся его реализацией. Для простоты далее будет описана только реализация протокола `zabbix_sender` – простейшего способа передачи данных в Zabbix.

Для описания объектов с данными в Zabbix используется формат JSON. Существует множество библиотек, обеспечивающих эффективную работу с этим форматом, но для простоты мы не будем использовать ни одну из них.

Реализация протокола Zabbix sender на Java

В этом разделе представлена простейшая из возможных реализация протокола `zabbix_sender`, который, как вы знаете, применяется для передачи данных в ловушки Zabbix.

Фрагмент кода, что приводится ниже, сохранен настолько простым, насколько это возможно, и наглядно демонстрирует, с чего начать, если вы задумаете реализовать собственный компонент мониторинга для системы Zabbix:

```
private String buildJSonString(String host, String item,
                               Long timestamp, String value)
{
  return "{"
    + "\"request\":\"sender data\",\n"
    + "\"data\":[\n"
    +          "{\n"
    +           "\"host\":\"" + host + "\",\n"
    +           "\"key\":\"" + item + "\",\n"
    +           "\"value\":\"" + value.replace("\\", "\\\\")
```

```
    +           "\",\n"
    +           "\"clock\":" + timestamp.toString()
    +         "}]}\n" ;
}
```

Функция buildJSonString просто возвращает текст в формате JSON для передачи в теле сообщения. Она требует передать ей только имя хоста, имя элемента (или, еще лучше, ключ элемента), значение и отметку времени для включения в сообщение и возвращает сформированную строку с объектом JSON.

Теперь, как только вы получите значения элементов, вы сможете сгенерировать сообщение в формате JSON, открыть соединение с сервером и послать это сообщение. Ниже показано, как можно открыть соединение с сервером Zabbix:

```
String data = buildJSonString(host,item,value);
zabbix = new Socket(zabbixServer, zabbixPort);
zabbix.setSoTimeout(TIMEOUT);
out = new OutputStreamWriter(zabbix.getOutputStream());
int length = data.length;
```

Этот код открывает сокет, определяет тайм-аут и определяет длину сообщения. После его выполнения все готово к отправке сообщения. Напомню, что сообщение включает три компонента: <ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ><ДАННЫЕ>. Ниже показан простейший способ отправки заголовка и числа, отражающего объем данных:

```
out.write(new byte[] {
  'Z', 'B', 'X', 'D',
  '\1',
  (byte)(length & 0xFF),
  (byte)((length >> 8) & 0x00FF),
  (byte)((length >> 16) & 0x0000FF),
  (byte)((length >> 24) & 0x000000FF),
  '\0','\0','\0','\0'});
```

Следующий фрагмент выводит в сокет сообщение, содержащее имя хоста, имя (ключ) элемента и значение:

```
out.write(data);
```

Теперь важно не забыть вытолкнуть буфер и закрыть сокет, завершив тем самым отправку:

```
out.flush();
out.close();
```

Далее – посмотреть, что скажет сервер Zabbix об отправленном элементе:

```
in = zabbix.getInputStream();
final int read = in.read(response);
String respStatus = (String) getValue(response);
if (read !=2 || respStatus.equals(ZBX_SUCCESS)) {
  in.close();
}
```

Если сервер сообщит об успехе, можно закрыть InputStream.

Это полностью действующий пример, но он служит исключительно учебным целям. Чтобы этот код можно было использовать для работы, в него необходимо внести еще несколько усовершенствований. Тем не менее он послужит отличной отправной точкой. Данный пример можно расширить для отправки сразу нескольких объектов JSON в разделе data, чтобы уменьшить количество устанавливаемых соединений с сервером. Вообще, старайтесь свести к минимуму число соединений, чтобы не перегружать сервер. Элементы можно сохранять в буфер и отправлять все сразу; например, если имеется группа элементов с одинаковым графиком отправки, их все можно отправить в одном сообщении.

Извлекая элементы, не забывайте добавлять в них отметки времени. Тогда вы будете знать, когда было извлечено то или иное значение.

В предыдущем примере отметка времени не посылается, так как она является необязательной, но вообще считается хорошим тоном включать ее в сообщение, особенно если вы предполагаете накапливать элементы в буфере перед отправкой.

Реализация протокола Zabbix sender на Python

В наши дни очень много приложений пишется на языке Python, получившем широкую известность и распространение. Поэтому в данном разделе мы рассмотрим начальную реализацию протокола zabbix_sender на Python. Вы сможете дополнить ее и встроить в свои программы. Интегрированная функциональность даст вашим приложениям очень интересную возможность сообщать серверу Zabbix о своем состоянии. Приступим.

Во-первых, необходимо определить структуру и импортировать модуль simplejson, используемый для добавления имени хоста, ключа, значения и отметки времени в формате JSON:

```
import simplejson as smplj
items_data = []
```

Получим отметку времени из элемента; если она отсутствует, определим текущее время:

```
clock = zbxit.clock or time.time()
```

Создадим объект JSON для включения в сообщение:

```
items_data.append(('\t\t{\n'
                   '\t\t\t"host":%s,\n'
                   '\t\t\t"key":%s,\n'
                   '\t\t\t"value":%s,\n'
                   '\t\t\t"clock":%s}')
                   %
                   (smplj.dump(zbxit.host),
                    smplj.dump(zbxit.key),
                    smplj.dump(zbxit.value), clock))
```

После преобразования элемента данных в объект JSON можно перейти к заголовку:

```
json_items = ('{\n'
              '\t"request":"sender data",\n'
              '\t"data":[\n%s]\n'
              '}')
              %
              (',\n'.join(items_data))
```

Следующий шаг – определение длины сообщения и добавление ее в заголовок:

```
data_len = struct.pack('<Q', len(json_items))
```

Как рассказывалось выше, сообщение имеет структуру: <ЗАГОЛОВОК><ОБЪЕМ_ДАННЫХ><ДАННЫЕ>:

```
packet = 'ZBXD\1' + data_len + json_items
```

Теперь откроем сокет и отправим сообщение:

```
zabbix = socket.socket()
zabbix.connect((zabbix_host, zabbix_port))
zabbix.sendall(packet)
```

После отправки пакета получим ответ сервера Zabbix:

```
resp_hdr = _recv_all(zabbix, 13)
```

Проверим отсутствие ошибок:

```
if not resp_hdr.startswith('ZBXD\1') or len(resp_hdr) != 13:
    return False
resp_body_size = struct.unpack('<Q', resp_hdr[5:])[0]
resp_body = zabbix.recv(resp_body_size)
zabbix.close()
resp = smplj.loads(resp_body)
if resp.get('response') != 'success':
    return False
return True
```

Этот код послужит вам отличной отправной точкой для собственной реализации протокола Zabbix sender на Python.

Некоторые замечания о разработке агента

Сейчас у вас, возможно, нет необходимости приступать к разработке своего программного обеспечения, посылающего данные серверу Zabbix. Тем не менее имейте в виду, что существуют определенные ограничения и требования к такого рода программному обеспечению.

В настоящий момент вы имеете два действующих примера, пусть не до конца проработанных, которые позволят вам посылать данные серверу Zabbix.

Обратите внимание, что отправка данных в этих примерах осуществляется без учета интервалов, установленных в настройках на сервере, и не может управляться сервером. Поэтому вы должны определить для себя, как будет осуществляться управление интервалами – вашей программой или сервером Zabbix. Чтобы дать серверу возможность управлять частотой измерения параметров, нужно реализовать протокол агента Zabbix agent. Протокол агента лишь немногим сложнее в реализации. Но, как бы то ни было, два представленных примера включают все компоненты, необходимые для реализации протокола агента.

Еще одно важное замечание: обычно разработчики имеют собственные предпочтения в выборе языка программирования. Поэтому важно уметь выбрать инструмент, подходящий для решения задачи. Представьте, например, что перед вами стоит задача осуществить мониторинг базы данных Oracle. В этом случае наиболее логичным выглядит выбор языка Java. Представляю, как сейчас фанаты Python сморщили носы! Но не торопитесь. Этот выбор продиктован не только и не столько личными предпочтениями. Помните, что для решения задач мониторинга лучше использовать проверенные и надежные инструменты.

Oracle и другие компании, разрабатывающие СУБД, поставляют также стандартные промышленные драйверы JDBC для Java, которые постоянно развиваются, дорабатываются, исправляются и улучшаются. Лучше делегировать часть работы этим компаниям. Кроме того, они знают свои продукты во всех тонкостях, и вы можете воспользоваться этими знаниями, воплощенными в драйверы.

Язык Java включает огромное количество хорошо проработанных компонентов, которые могут помочь вам в решении сложной задачи мониторинга базы данных. Например, фреймворк JDBC вместе с драйвером базы данных обеспечит вас надежным пулом соединений, в котором можно настроить:

- минимальное и максимальное количество соединений;
- проверку соединения перед использованием программой;
- отправку пакетов, поддерживающих соединение открытым (удобно для преодоления проблем с брандмауэрами);
- время удержания соединения открытым, после которого все неактивные соединения закрываются (помогает уменьшить общее количество соединений с подконтрольным сервером).

Это лишь несколько пунктов из длинного списка достоинств JDBC. Все эти достоинства помогут вам сохранить реализацию мониторинга простой, легковесной и эффективной.

Примером обобщенного программного обеспечения для мониторинга баз данных может служить DBforBIX, доступный по адресу: http://sourceforge.net/projects/dbforbix/ или http://www.smartmarmot.com/product/dbforbix/.

В заключение

В этой главе были представлены все возможные методы взаимодействий с сервером Zabbix, позволяющие организовать сбор данных, недоступных другими спо-

собами. Здесь мы рассмотрели шаги, которые необходимо выполнить, чтобы перенести сценарий мониторинга Oracle со стороны сервера на сторону агента, а затем на выделенный сервер. Вы узнали, как простой сценарий постепенно превращается в сложную программу. На каждом шаге мы с вами исследовали и обсудили достоинства и недостатки каждого из местоположений сценария. У вас не должно складываться впечатление, что все внешние проверки следует осуществлять с применением выделенного сервера; этот подход оправдан, только если имеется большое количество интенсивно используемых проверок. Теперь вы имеете достаточно полное представление, чтобы обоснованно выбрать наиболее подходящее место, где должны храниться и действовать сценарии. Кроме того, мы раскрыли все секреты протоколов Zabbix, и теперь вы сможете расширять систему мониторинга на основе, не зная ограничений.

В следующей главе вы узнаете, как взаимодействовать с Zabbix посредством прикладного программного интерфейса (API). Она познакомит вас с преимуществами использования Zabbix API для массового развертывания хостов и пользователей и массового выполнения повторяющихся операций вообще.

Глава 9

Расширение Zabbix

Знание протоколов мониторинга Zabbix позволяет писать сценарии, агентов и осуществлять отправку данных. Иными словами, это знание позволяет расширять возможности мониторинга Zabbix, добавляя новые инструменты сбора данных.

Однако для администрирования объектов мониторинга мы пока можем использовать только веб-интерфейс. Чтобы добавить пользователя, изменить частоту опроса элемента или просмотреть исторические данные, мы должны использовать веб-интерфейс.

Это, конечно, очень удобный инструмент для решения повседневных задач, так как для его использования достаточно иметь веб-браузер. Сам по себе веб-интерфейс – достаточно мощное и гибкое средство, позволяющее к тому же выполнять массовые операции со множеством объектов одного типа и управлять разными прокси-серверами из одного места.

Но не каждая массовая и сложная операция может быть выполнена с помощью веб-приложения, и иногда требуется не просто визуализировать данные, а экспортировать их и передать другой программе для дополнительной обработки. В таких случаях на помощь к нам приходит Zabbix API. Как вы узнаете в этой главе, прикладной интерфейс Zabbix JSON-RPC API предоставляет все функции, доступные через веб-интерфейс, включая управление пользователями и конфигурацией мониторинга, а также доступ к историческим данным.

Эта глава охватывает следующие темы:

- программное подключение к API и выполнение запросов к нему;
- создание пользовательских операций для управления конфигурацией;
- создание сложных массовых операций с условиями;
- экспортирование данных мониторинга в разных форматах.

Начнем с общего знакомства с архитектурой прикладного интерфейса и особенностями разработки программного кода для взаимодействия с ним.

Zabbix API

Сервер Zabbix предоставляет точку входа для взаимодействий с ним, создания и настройки объектов и управления ими. Этот прикладной программный интерфейс доступен через интерфейс PHP по адресу: `http://<сервер_zabbix>/zabbix/api_jsonrpc.php`.

Протокол взаимодействий основан на применении формата JSON, а роль носителя, как обычно, играют протоколы HTTP/HTTPS.

Zabbix JSON-RPC API – отличный интерфейс, открывающий доступ к большому количеству функций. После аутентификации он позволит вам выполнять самые разные операции с объектами Zabbix. Теперь, если вам понадобится развернуть Zabbix в большой или очень большой сети, Zabbix API придется как нельзя кстати. Например, представьте, что вам потребовалось добавить большое количество устройств, которые уже перечислены в отдельном документе. Используя API, вы сможете добавить их в Zabbix, написав простой сценарий.

Впервые Zabbix API был реализован в версии Zabbix 1.8 и продолжал активно изменяться до версии 2.4. Эту версию можно считать самой стабильной и надежной, но API продолжает развиваться и изменяться, поэтому в более новых версиях кое-что может не соответствовать описанию в этой главе. Это не означает, что API нельзя использовать в промышленном окружении; напротив, чем больше окружение, тем выгоднее применять API для выполнения сложных и продолжительных операций.

Ниже представлен упрощенный JSON-запрос к Zabbix API:

```
{
  "jsonrpc": "2.0",
  "method": "method.name",
  "params": {
    "param_1_name": "param_1_value",
    "param_2_name": "param_2_value"
  },
  "id": 1,
  "auth": "159121ba47d19a9b4b55124eab31f2b81"
}
```

Пункты следующего списка поясняют назначение строк в этом примере:

- `"jsonrpc": "2.0"`: стандартный параметр JSON PRC, используемый для идентификации версии протокола; эта строка без изменений используется во всех запросах;
- `"method": "method.name"`: этот параметр определяет операцию для выполнения, например `host.create` или `item.update`;
- `"params"`: определяет блок параметров, необходимых методу JSON, например если понадобится создать элемент данных, в этом блоке можно передать параметры `"name"` и `"key_"`;
- `"id": 1`: это поле используется для связывания ответа с запросом; каждый ответ будет иметь тот же идентификатор `"id"`, что и соответствующий ему запрос; поле `"id"` удобно использовать в случаях, когда требуется послать сразу несколько запросов;
- `"auth": "159121ba47d19a9b4b55124eab31f2b81"`: ключ (токен) аутентификации, используемый для идентификации аутентифицированного пользователя; подробнее об этом параметре рассказывается в следующем разделе.

Подробное описание всех возможных параметров и методов можно найти в официальной документации: https://www.zabbix.com/documentation/2.4/ru/manual/api/reference.

Также важно понимать, что все взаимодействия осуществляются по протоколу HTTP. Это необходимо учитывать, взаимодействуя с Zabbix со своей рабочей станции или из другого места в сети. Для взаимодействия с Zabbix API прежде всего необходимо пройти процедуру аутентификации на сервере, соответственно, очень важно, чтобы передача всех данных производилась с шифрованием. В этом отношении есть две уязвимости:

- при использовании `http` вместо `https` учетные данные передаются по сети в открытом виде и могут быть перехвачены;
- некоторые данные могут составлять конфиденциальную информацию.

Теперь настал момент сделать первый шаг к использованию API – запросить версию API после аутентификации.

Первые шаги

Прежде всего, поскольку для взаимодействий с Zabbix API используются HTTP-запросы `POST`, в следующих примерах мы будем использовать утилиту `curl`, чтобы лучше понять протокол.

С помощью `curl` можно легко и быстро передавать и получать данные от служб, используя разные протоколы, и здесь, в первом примере, мы используем HTTP. Даже при том, что он незащищенный, это не является проблемой, так как мы просто запросим версию Zabbix и не будем выполнять аутентификацию или передавать конфиденциальные данные.

```
$ curl --include --netrc --request POST --header "Content-Type: application/json"
http://127.0.0.1/zabbix/api_jsonrpc.php -d@-
```

В числе параметров мы определили заголовок `Content-Type`, где указали, что запрос содержит данные в формате JSON, и потребовали от `curl` принять данные со стандартного ввода, добавив ключ `-d@-`. Вслед за этой командой введите следующие строки:

```
{
"jsonrpc":"2.0",
"method":"apiinfo.version",
"id":1,
"auth":null,
"params":{}
}
```

и завершите ввод комбинацией **Crtl+D**.

Теперь рассмотрим ответ:

```
HTTP/1.1 200 OK
Date: Sat, 04 Jul 2015 06:32:36 GMT
Server: Apache/2.2.15 (CentOS)
```

```
X-Powered-By: PHP/5.3.3
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Content-Type
Access-Control-Allow-Methods: POST
Access-Control-Max-Age: 1000
Content-Length: 41
Connection: close
Content-Type: application/json

{"jsonrpc":"2.0","result":"2.4.5","id":1}
```

Вслед за стандартным HTTP-заголовком ответа выводится результат запроса, то есть версия Zabbix: `"result":"2.4.5"`.

Имейте в виду, что метод `apiinfo.version` появился только в версии Zabbix 2.0.4. В более старых версиях Zabbix этот метод не поддерживается.

Аутентификация

В этом разделе мы рассмотрим более подробный пример, демонстрирующий, насколько просто осуществляются взаимодействия, а затем обсудим его реализацию на языке Python, который широко используется для быстрой разработки приложений.

Для тестирования процедуры аутентификации из командной оболочки вновь воспользуемся утилитой `curl`. Поскольку на этот раз речь идет об аутентификации, важно использовать защищенный протокол `https`. Итак, зарегистрироваться на сервере Zabbix можно с помощью следующей команды:

```
$ curl --insecure --include --netrc --request POST --header "Content-Type: application/
json" https://127.0.0.1/zabbix/api_jsonrpc.php -d@-
```

Обратите внимание на параметр `--insecure`; он сообщает утилите `curl`, что она не должна проверять сертификат сервера. Это несколько снижает защищенность, но, поскольку мы работаем с локальным сервером, это вполне допустимо и помогает избежать проблем с сертификатами. Если выполнить эту команду без параметра `--insecure`, `curl` выведет следующее сообщение об ошибке:

```
curl: (60) Peer certificate cannot be authenticated with known CA certificates
More details here: http://curl.haxx.se/docs/sslcerts.html
```

Вслед за этой командой введите следующие строки:

```
{
  "jsonrpc": "2.0",
  "method": "user.login",
  "params": {
    "user": "Admin",
    "password": "my secret password"
  },
  "auth": null,
  "id": 0
}
```

Не забудьте подставить свой пароль в параметр "password" и завершить ввод комбинацией клавиш **Crtl+D**.

Утилита curl позаботится о создании соединения HTTPS и вернет полный HTTP-ответ сервера. В данном случае нас интересует ключ (токен) аутентификации, следующий за стандартным HTTP-заголовком:

```
HTTP/1.1 200 OK
Content-Type: application/json
{"jsonrpc":"2.0","result":"403bbcdc3c01d4d6e66f68f5f3057c3a","id":0}
```

Этот ответ содержит ключ, который мы должны будем использовать во всех последующих запросах к серверу Zabbix.

Действие ключа истекает через время, указанное в параметре настройки autologout.

Теперь попробуем получить данные, для чего снова воспользуемся утилитой curl:

```
# curl --insecure --include --netrc -request POST --header "Content-Type: application/json"
https://127.0.0.1/zabbix/api_jsonrpc.php -d @-
```

В этом примере мы запрашиваем у сервера последнее историческое значение для элемента Processor load (15 min average per core). В данном конкретном случае для данного сервера тело запроса имеет следующий вид:

```
{ "jsonrpc": "2.0",
  "method": "history.get",
  "params": {
    "output": "extend",
    "history": 0,
    "hostids": "10096",
    "itemid": "23966",
    "sortfield": "clock",
    "sortorder": "DESC",
    "limit": 1
  },
  "auth": "403bbcdc3c01d4d6e66f68f5f3057c3a",
  "id": 1
}
```

Не забывайте, что запрос должен содержать ключ аутентификации, полученный прежде обращением к методу "user.authenticate".

В большинстве своем API содержат как минимум четыре метода: get, create, update и delete, но имейте в виду, что некоторые могут поддерживать совершенно другой набор методов.

В данном случае сервер вернул следующий ответ:

```
HTTP/1.1 200 OK

{"jsonrpc":"2.0",
```

```
  "result":[
  {
    "hosts": [{"hostid":"10096"}],
    "itemid":"23840",
    "clock":"1381263380",
    "value":"0.1506",
    "ns":"451462502"}
  ],
  "id":1
}
```

Благодаря этому примеру мы узнали, как использовать ключ аутентификации для запроса исторических данных. Конечно, командная оболочка – не самый лучший инструмент для взаимодействий с Zabbix API, потому что требует определенных усилий для управления ключом `"auth"`, и было бы предпочтительнее использовать что-то более дружественное.

Использование библиотеки PyZabbix

Теперь, когда вы получили представление об архитектуре API и протоколе JSON-RPC, можно выйти за рамки конструирования объектов JSON вручную и включить в работу специализированную библиотеку. Это позволит сосредоточиться на фактических возможностях API, а не на особенностях его реализации.

Существует несколько версий библиотеки Zabbix API на разных языках, и в оставшейся части главы мы будем использовать одну из них – PyZabbix, написанную Люком Сайке (Luke Cyca) (https://github.com/lukecyca/pyzabbix/wiki). Это небольшой, компактный модуль Python, который достаточно близок к оригинальному API и прост в использовании. Кроме того, интерактивная оболочка Python дает удобную возможность опробовать возможности библиотеки и сконструировать прототип, прежде чем приступать к созданию законченного сценария или приложения.

Установить библиотеку PyZabbix можно с помощью Pip – диспетчера пакетов Python:

```
$ pip install pyzabbix
```

После установки модуля его можно импортировать и использовать в сценариях, управляющих инфраструктурой Zabbix.

Сначала создадим объект для доступа к API сервера и получим ключ аутентификации.

Ниже приводится фрагмент сеанса в интерактивной оболочке Python, но этот же код можно использовать в сценариях:

```
>>> from pyzabbix import ZabbixAPI
>>> zh = ZabbixAPI("https://127.0.0.1/zabbix/")
>>> zh.login("Admin", "zabbix")
```

Разумеется, вы должны подставить в этот код фактический URL веб-интерфейса Zabbix и свои учетные данные. Если все сделано правильно, аутентификация

пройдет как по маслу. После этого вы сможете использовать объект zh для доступа к любым методам API:

```
>>> zh.host.get(output="refer")
```

Для параметра "refer" метод get возвращает первичный и внешние ключи объекта:

```
[{'hostid': '9909900000010084'},
 {'hostid': '9909900000010085'},
 {'hostid': '9909900000010086'},
 {'hostid': '9909900000010087'},
 {'hostid': '9909900000010088'}]
```

Другое преимущество библиотеки PyZabbix заключается в прозрачном преобразовании объектов JSON в объекты Python, что практически полностью избавляет от необходимости выполнять преобразования типов. В табл. 9.1 перечислены конкретные типы, поддерживаемые Zabbix API, и примеры, как они выглядят в формате JSON и в вызовах функций PyZabbix.

Таблица 9.1 ❖ Соответствие типов Zabbix API, JSON и PyZabbix

Тип	JSON	PyZabbix

bool	{
	"jsonrpc" : "2.0"
	"method": "host.get",
	"params" : {
	"editable" : "**true**"
	},
	"auth" : <....>
	"id" : 1
	}
		zh.host.get(editable="**true**")

flag	{
	"jsonrpc" : "2.0"
	"method": "host.get",
	"params" : {
	"countOutput" : "**1**"
	},
	"auth" : <....>
	"id" : 1
	}
		zh.host.get(countOutput=**1**)

integer	{
	"jsonrpc" : "2.0"
	"method": "host.get",
	"params" : {
	"limit" : **10**
	},
	"auth" : <....>
	"id" : 1
	}
		zh.host.get(limit=**10**)

Таблица 9.1 *(окончание)*

Тип	JSON	PyZabbix
string	{ "jsonrpc" : "2.0" "method": "host.get", "params" : { "sortfield": "name" }, "auth" : <....> "id" : 1 }	zh.host.get(sortfield="name")
timestamp	{ "jsonrpc": "2.0", "method": "event.get", "params": { "time_from": "1349797228", "time_till": "1350661228", }, "auth": <...>, "id": 1 }	zh.event.get(time_from="1349797228", time_till="1350661228")
array	{ "jsonrpc" : "2.0" "method": "host.get", "params" : { "hostids" : [1001, 1002, 1003] }, "auth" : <....> "id" : 1 }	zh.host.get(hostids=[1001, 1002, 1003])
object	{ "jsonrpc" : "2.0" "method": "host.get", "params" : { "filter": { "name": ["Alpha", "Beta"] } }, "auth" : <....> "id" : 1 }	zh.host.get(filter={"name": ["Alpha", "Beta"]})
query	{ "jsonrpc" : "2.0" "method": "host.get", "params" : { "output": "extend" }, "auth" : <....> "id" : 1 }	zh.host.get(output="extend")

Библиотека конструирует запросы «на лету», поэтому она автоматически будет поддерживать любые методы API, которые появятся в будущем.

Теперь вы можете заняться исследованием конкретных примеров использования API. Чтобы облегчить чтение кода и помочь сосредоточиться на особенностях API, а не на проблемах программирования, все примеры выше были максимально упрощены, и в них отсутствуют проверки ошибок и допустимости данных. Представленные фрагменты можно опробовать в интерактивной оболочке Python или использовать в своих приложениях (или даже создать на их основе комплект инструментов командной строки), но я настоятельно рекомендую доработать их, добавив обработку ошибок и проверку допустимости данных.

Исследование Zabbix API с помощью JQuery

Еще один интересный проект, который вы можете загрузить и опробовать: JQZabbix. Подробное описание проекта можно найти по адресу https://github.com/kodai/jqzabbix.

JQZabbix – это демонстрационное приложение, позволяющее исследовать Zabbix API. Его даже можно установить где-нибудь в действующем окружении и использовать для отладки, так как оно позволяет с помощью обычного браузера выполнять запросы JSON-RPC без необходимости писать сценарии.

Чтобы установить пакет, его требуется загрузить; с этой целью можно просто клонировать репозиторий GitHub:

```
$ mkdir jqzabbix && cd jqzabbix
$ git clone https://github.com/kodai/jqzabbix
```

Эти команды создадут копию репозитория проекта в каталоге `jqzabbix`. Теперь нужно создать новую папку `jqzabbix` в корневом каталоге документов веб-сервера. Обычно роль такого корневого каталога играет `/var/www/html`, но вообще лучше проверить значение директивы `DocumentRoot` в конфигурационном файле `/etc/httpd/conf/httpd.conf`. Выполните следующие команды от имени root, чтобы скопировать файлы проекта:

```
$ mkdir /var/www/html/jqzabbix
$ cp <your-jqzabbix-location>/demo/* /var/www/html/jqzabbix/
$ cp <your-jqzabbix-location>/jqzabbix/* /var/www/html/jqzabbix/
```

Теперь откройте в редакторе файл `main.js` и измените следующую строку:

```
// Zabbix server API url
var url = 'http://localhost/zabbix/api_jsonrpc.php';
```

Переменная `url` должна содержать фактический IP-адрес или доменное имя сервера Zabbix.

После этого запустите браузер и откройте в нем страницу `http://<zabbix-server>/jqzabbix/`. На экране должна появиться страница, как показано на рис. 9.1.

Это приложение интересно тем, что представляет собой очень наглядный пример использования Zabbix API с применением библиотеки jQuery. Данное приложение позволяет использовать практически все методы, поддерживаемые Zabbix API (рис. 9.2).

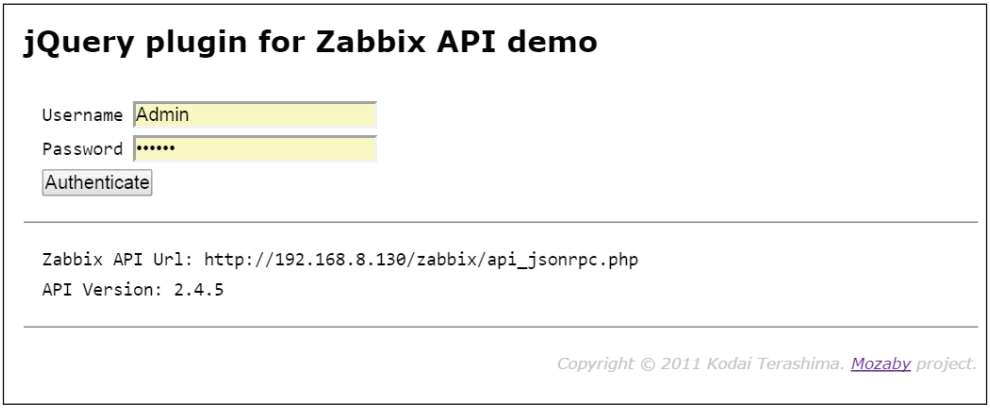


Рис. 9.1 ❖ Начальная страница веб-приложения JQZabbix

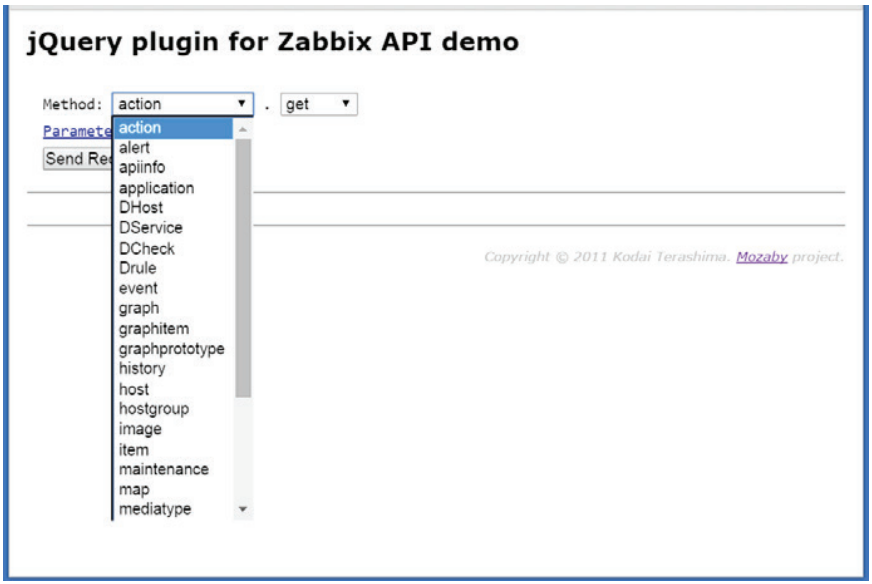
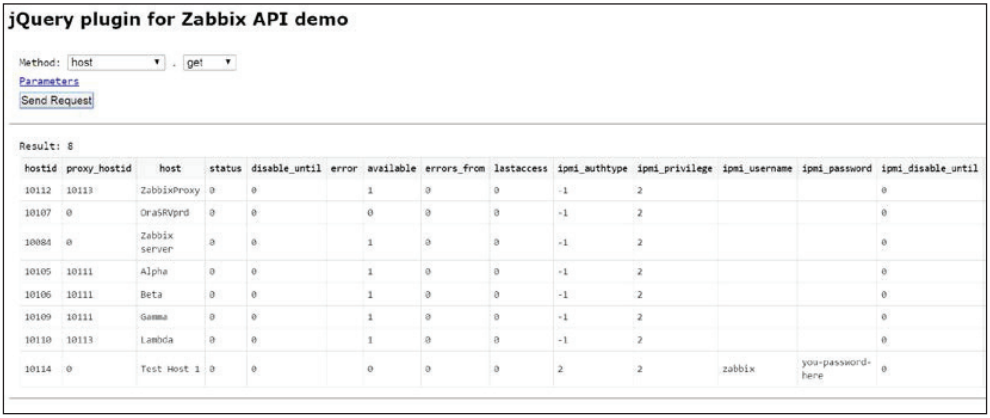


Рис. 9.2 ❖ JQZabbix позволяет использовать практически все методы Zabbix API

Например, на рис. 9.3 показан результат вызова метода `host.get`.

Давайте внимательнее посмотрим, как действует это приложение. Вы можете открыть в редакторе файл `main.js`, чтобы просматривать программный код параллельно с чтением книги. В первую очередь здесь создается объект `jqzabbix`. Параметры конструктора по большей части необязательные, так как для них заданы следующие значения по умолчанию:

```
server = new $.jqzabbix({
  // URL для доступа Zabbix API
  url: 'http://localhost/zabbix/api_jsonrpc.php',
  username: 'Admin',  // имя пользователя Zabbix
  password: 'zabbix', // пароль
  basicauth: false,   // если используется метод
                      // аутентификации BASIC, передайте в
                      // этом параметре значение true
  busername: '',      // Имя пользователя для аутентификации
                      // методом BASIC
  bpassword: '',      // Пароль для аутентификации
                      // методом BASIC
  timeout: 5000,      // Тайм-аут запроса (в миллисекундах)
  limit: 1000,        // Максимальное количество данных на запрос
})
```



hostid	proxy_hostid	host	status	disable_until	error	available	errors_from	lastaccess	ipmi_authtype	ipmi_privilege	ipmi_username	ipmi_password	ipmi_disable_until
10112	10113	ZabbixProxy	0	0		1	0	0	-1	2			0
10107	0	OraSRVprd	0	0		0	0	0	-1	2			0
10084	0	Zabbix server	0	0		1	0	0	-1	2			0
10105	10111	Alpha	0	0		1	0	0	-1	2			0
10106	10111	Beta	0	0		1	0	0	-1	2			0
10109	10111	Gamma	0	0		1	0	0	-1	2			0
10110	10113	Lambda	0	0		1	0	0	-1	2			0
10114	0	Test Host 1	0	0		0	0	0	2	2	zabbix	you-password-here	0

Рис. 9.3 ❖ Результат вызова метода `host.get`

Затем запрашивается версия Zabbix API:

```
server.getApiVersion();
```

Если запрос завершился успехом, выполняется аутентификация:

```
server.userLogin();
```

В случае успешной аутентификации ключ (токен) сохраняется в свойстве объекта `server`. После этого можно вызвать обычный метод API:

```
server.sendAjaxRequest(method, params, success, error)
```

Параметры этого метода интерпретируются следующим образом:

- `method`: метод Zabbix API из перечисленных в описании Zabbix API;
- `params`: параметры метода Zabbix API;
- `success`: функция обратного вызова, которая должна быть выполнена в случае успеха;

- `error`: функция обратного вызова, которая должна быть выполнена в случае ошибки.

Как видите, это очень простой пакет, но очень удобный, например, для сравнения значений с целью убедиться, что ваши сценарии работают правильно. Плюс это отличная начальная точка для желающих написать свое приложение, использующее библиотеку jQuery. Zabbix API открывает перед нами почти безграничные возможности, и спасибо разработчикам API, что дали нам средства автоматизации многих рутинных задач.

Массовые операции

Теперь настал момент рассмотреть Python Zabbix API в действии. Еще одна типичная область применения API – автоматизация некоторых операций, доступных через веб-интерфейс, но слишком утомительных и чреватых ошибками. К таким операциям относятся, например, добавление большого количества пользователей или изменение IP-адресов хостов после слияния двух сетей. В следующих фрагментах кода предполагается, что соединение с Zabbix API уже выполнено, как было показано в предыдущем разделе. Иными словами, с этого момента предполагается, что код начинается со строк, представленных ниже (не забывайте, что адрес URL и учетные данные пользователя здесь приведены для примера, и вместо них вы должны подставить свои значения!):

```
#!/usr/bin/python
from pyzabbix import ZabbixAPI
user='Admin'
pwd='password'
url = 'https://127.0.0.1/zabbix/'
zh = ZabbixAPI(url)
zh.login(user=user, password=pwd)
```

Перераспределение хостов между прокси-серверами

Мы видели в *главе 2 «Распределенный мониторинг»*, что связывать хосты с прокси-сервером можно в форме настройки прокси-сервера или в форме настройки каждого хоста, определяя прокси в параметре `monitored by`. Оба способа могут быть утомительными и требовать немало времени при большом количестве хостов, настройки которых требуется изменить. Если нужно просто передать группу хостов от одного прокси-сервера другому, можно воспользоваться функцией массового обновления, поддерживаемой веб-интерфейсом, но если необходимо распределить хосты между разными прокси-серверами или изменить настройки лишь нескольких хостов из разных групп, такой подход малопригоден.

Далее демонстрируется пример перераспределения всех хостов, связанных с некоторым прокси-сервером, между остальными прокси в инфраструктуре Zabbix. Это может понадобиться, например, чтобы остановить некоторый прокси-сервер для обслуживания, без прекращения мониторинга группы хостов.

Во-первых, определим идентификатор и имя прокси:

```
proxy_name = "ZBX Proxy 1"
proxy_id = zh.proxy.get(
    filter={"host": proxy_name}, output="refer")[0]['proxyid']
```

После этого получим список хостов, мониторинг которых осуществляется этим прокси-сервером:

```
hlist = zh.proxy.get(selectHosts=['hostid'], proxyids=proxy_id,
    output="refer")[0]['hosts']
hosts = [x['hostid'] for x in hlist]
```

Далее, для упрощения примера, получим список всех остальных прокси, исключив тот, что мы собираемся остановить:

```
proxies = [x['proxyid'] for x in zh.proxy.get()
                        if x['proxyid'] != proxyid]
```

Теперь разобьем список переподчиняемых хостов на примерно одинаковые группы, по количеству доступных прокси-серверов:

```
nparts = int(round(len(hosts)/len(proxies)))
hostchunks = [list(hosts[i:i+nparts])
              for i in range(0,len(hosts),nparts)]
```

Предыдущий фрагмент кода разделит список хостов на множество мелких списков по количеству доступных прокси-серверов. Теперь осталось только связать хосты с прокси-серверами:

```
for c in len(hostchunks):
    zh.proxy.update(proxyid=proxies[c], hosts=hostchunks[c])
```

Вот и все. Метод `proxy.update` автоматически выполнит перепривязку хостов, поэтому вам даже не придется предварительно отвязывать их от прокси-сервера. Вы можете улучшить эту реализацию, например отбирать прокси-серверы из той же сети, что и останавливаемый на обслуживание, или сохранять список хостов, чтобы вновь подключить их к своему прокси после его запуска.

Добавление и изменение учетных записей

Даже если аутентификация пользователей Zabbix выполняется с применением внешних механизмов, таких как сервер LDAP или Active Directory, в новых учетных записях все равно не определяется информация о способах оповещения или о принадлежности к каким-то группам. Это означает, что каждую учетную запись вам придется настроить вручную, если вы не воспользуетесь каким-то программным кодом. Ради простоты предположим, что у нас уже имеется список имен пользователей, адресов электронной почты и названий групп, которым они принадлежат. Вся информация собрана в файле `users.csv`, который выглядит, как показано ниже:

```
adallevacche,a.dallevacche@example.com,Admins
jdoe,jdoe@foo.bar,DB admins; App Servers
mbrown,mbrown@example.org,Net admins
```

Первое поле в каждой строке содержит имя пользователя (на языке Zabbix API называется `alias`). Второе поле содержит адрес электронной почты, а последнее – список групп, перечисленных через точку с запятой, которым должен принадлежать данный пользователь.

Изменение существующей учетной записи выполняется просто. Сначала нужно прочитать содержимое файла `users.csv`:

```
with open('users.csv', 'r') as f:
    users = f.readlines()
```

С помощью объекта соединения с Zabbix API (допустим, что он называется `zh`) теперь можно определить пару вспомогательных функций и переменных. Переменная `mediatypeid` потребуется для изменения методов доставки оповещений пользователям. Допустим, что у нас поддерживается только способ оповещения по электронной почте, и вы можете получить его идентификатор:

```
mediatypeid = zh.mediatype.get(output="refer",
    filter={"description": ['Email']})[0]['mediatypeid']
```

Если только вы не собираетесь дополнить свой файл `.csv` информацией о важности и периодах времени, когда рассылка оповещений допустима, для каждого способа можно также определить общий шаблон для контактов по электронной почте:

```
def mkusermedia(mediatype='', email='', mediaid=''):
    return { "mediaid": mediaid
             "mediatypeid": mediatype,
             "sendto": email,
             "active": 0,
             "severity": 63,
             "period": "1-7,00:00-24:00"
    }
```

Обратите внимание, что значение `0` в свойстве `active` означает *включено*, а `1` – *выключено*. Свойство `period` выглядит достаточно очевидным, а вот свойство `severity` на первый взгляд кажется туманным. Фактически это двоичная маска со значениями важности триггеров – каждому уровню важности соответствует свой бит в маске, как показано в табл. 9.2.

Таблица 9.2 ❖ Значение свойства severity = 63

Важность	Чрезвычайная	Высокая	Средняя	Предупреждение	Информация	Не классифицировано
Включено?	1	1	1	1	1	1
Десятичное значение	11111 = 63					

Так как число *63* имеет двоичный вид *111111*, это означает, что пользователь будет принимать оповещения любой важности. Если пользователь должен получать только чрезвычайные оповещения, свойству `severity` следует присвоить двоичное значение *100000*, или десятичное *32*, как показано в табл. 9.3.

Таблица 9.3 ❖ Значение свойства severity для рассылки только чрезвычайных оповещений

Важность	Чрезвычайная	Высокая	Средняя	Предупреждение	Информация	Не классифицировано
Включено?	1	0	0	0	0	0
Десятичное значение	10000 = 32					

Аналогично, чтобы организовать отправку оповещений с чрезвычайным и высоким уровнями важности, нужно свойству `severity` присвоить двоичное значение *110000*, или десятичное *48* (табл. 9.4), и т. д.

Таблица 9.4 ❖ Значение свойства severity для рассылки оповещений с чрезвычайным и высоким уровнями важности

Важность	Чрезвычайная	Высокая	Средняя	Предупреждение	Информация	Не классифицировано
Включено?	1	1	0	0	0	0
Десятичное значение	11000 = 48					

Следующая вспомогательная функция возвращает список существующих идентификаторов групп пользователей:

```
def getgroupids(grouplist):
    return zh.usergroup.get(output=['usergrpid'],
        filter={"name": grouplist.split(",")})
```

Теперь продолжим работу со списком учетных записей:

```
for u in users:
    (alias, email, groups) = u.split(",")
    user = zh.user.get(output=['userid'],
                       filter={"alias": [alias]})
    if not user:
        zh.user.create(alias=alias,
                       passwd="12345",
                       usrgrps=getgroupids(groups),
                       user_medias=[mkusermedia(
                                       mediatype=mediatypeid,
                                       email=email)])
```

Инструкция `if` проверяет, существует ли учетная запись. Если не существует, вызывается метод `user.create`, который создаст ее, добавит в указанные группы

и создаст контакт для передачи оповещений. Пароль обязательно должен быть определен, даже если аутентификация пользователей выполняется с помощью внешней службы. В зависимости от политики управления паролями следует принуждать пользователей изменить установленный по умолчанию пароль или, что еще лучше, вместо фиксированной динамически генерировать случайную последовательность символов для свойства `password`.

Во второй ветви инструкции `if` извлекается идентификатор `userid` и обновляется информация в учетной записи:

```
else:
    userid=user[0]['userid']
    zh.user.update(userid=userid,srgrps=getgroupids(groups))
    usermedia = zh.usermedia.get(filter={"userid" : userid},
                                 output=['mediaid'])
    zh.user.updatemedia(users = [userid],
                        medias=[mkusermedia(
                            mediaid=usermedia[0]['mediaid'],
                            mediatype=mediatypeid,
                            email=email)])
```

Обратите внимание, что здесь требуется вызвать два метода – для изменения группы пользователя и способа оповещения, – а не один. Первый обновит информацию о принадлежности пользователя к группе; второй проверит адрес электронной почты и обновит или создаст его.

Предыдущий код можно выполнять периодически, чтобы обеспечить своевременное обновление учетных записей. Дополнительно здесь можно было бы обновлять имя и фамилию пользователя, или запрашивать данные непосредственно у сервера LDAP, или извлекать их из любого другого источника, а не из файла `.csv`.

Экспортирование данных

Помимо мониторинга и управления внутренними объектами, Zabbix API можно также использовать для извлечения данных с целью дальнейшего анализа за границами веб-интерфейса Zabbix. Карты, экраны, графики, триггеры и таблицы с историческими данными являются отличными инструментами для составления отчетов, но они доступны только внутри веб-интерфейса. Иногда может понадобиться извлечь исходные данные, чтобы затем выполнить некоторые вычисления (например, для планирования расширения мощностей) или создать документ с собственными графиками и таблицами. Если такое необходимо постоянно, имеет смысл написать сценарий, который будет извлекать данные посредством API. Интересные возможности открывают методы `get`, считающиеся основными строительными блоками любого сценария, предназначенного для извлечения данных, – они изначально поддерживают разнообразные фильтры и параметры. Потратив немного времени на их изучение, вы обнаружите, что они позволяют добиться желаемого результата небольшого количества ясного и понятного кода, потому что

избавляют от необходимости организовывать собственную фильтрацию, но для этого вам понадобится конструировать точные и сосредоточенные запросы.

Давайте рассмотрим несколько коротких примеров.

Извлечение табличных данных

Zabbix поддерживает группировку схожих элементов данных хоста с целью упрощения навигации по ним при просмотре последних данных мониторинга. Эти контейнеры для элементов, называемые приложениями, особенно удобны, когда набор элементов для разных хостов практически не изменяется. Если сгруппировать все параметры работы центрального процессора, например под меткой **CPU**, все параметры файловых систем – под меткой **Filesystems** и т. д., их значительно проще будет отыскивать. Приложения в Zabbix – это всего лишь метки, связанные с определенным шаблоном или хостом, и используются они исключительно для классификации элементов и не применяются нигде больше в системе Zabbix.

Иногда состояния триггеров или историю событий полезно анализировать не только по хостам, но и по приложениям. Отчет обо всех проблемах, возникавших в работе сети, независимо от хостов, групп хостов или определенных триггеров, может пригодиться отдельным группам в департаменте информационных технологий. То же самое относится к отчетам о состоянии файловых систем, баз данных и т. д.

Давайте посмотрим, как написать сценарий, экспортирующий все события в файл `.csv`, связанные с определенным приложением. Подготовительные операции в этом сценарии практически те же, что и в предыдущих примерах:

```
#!/usr/bin/python
from pyzabbix import ZabbixAPI
import sys
import csv
from datetime import datetime

appname = sys.argv[1]
timeformat="%d/%m/%y %H:%M"
zh = ZabbixAPI("http://locahost/zabbix")
zh.login(user="Admin", password="zabbix")
```

Как видите, имя приложения извлекается из аргумента командной строки, а адрес URL API и учетные данные в данном случае служат лишь примерами. Для большей гибкости их можно определить в конфигурационном файле. Поскольку события записываются с использованием отметок времени в формате Unix, эти отметки позднее желательно преобразовать в строковое представление. Переменная `timeformat` позволяет определить предпочитаемый формат. Что касается форматов, модуль `csv` позволяет определять формат вывода более гибко, чем последовательность инструкций `print`.

Теперь извлечем все приложения с именами, совпадающими с указанным в командной строке:

```
applications = zh.application.get(output="shorten",
                                  filter={"name": [appname]})
```

Затем можно извлечь список элементов, принадлежащих указанному приложению:

```
items = zh.item.get(output="count",
    applicationids=[x['applicationid'] for x in applications])
```

список всех триггеров, содержащих элементы из списка `items`:

```
triggers = zh.trigger.get(output="refer",
    itemids=[x['itemid'] for x in items])
```

и, наконец, список событий, связанных с указанным приложением:

```
events = zh.event.get(
    triggerids=[j['triggerid'] for j in triggers])
```

Здесь извлекаются только идентификаторы событий. Мы не указали период времени, поэтому список событий может получиться очень большим. Для каждого события извлечем также соответствующие хосты, триггеры и элементы. Но сначала определим несколько вспомогательных функций, возвращающих имена хостов, элементов и триггеров:

```
def gethostname(hostid=''):
    return zh.host.get(
        hostids=hostid, output=['host'])[0]['host']

def getitemname(itemid=''):
    return zh.item.get(
        itemids=itemid, output=['name'])[0]['name']

def gettriggername(triggerid=''):
    return zh.trigger.get(
        triggerids=triggerid, expandDescription="1",
        output=['description'])[0]['description']
```

Наконец, определим пустую таблицу `eventstable` и заполним ее информацией о событиях, извлеченной прежде:

```
eventstable = []
triggervalues = ['OK', 'problem', 'unknown']
for e in events:
    eventid = e['eventid']
    event = zh.event.get(eventids=eventid,
                         selectHosts="refer",
                         selectItems="refer",
                         selectTriggers="refer",
                         output="extend")
    host = gethostname(event[0]['hosts'][0]['hostid'])
    item = getitemname(event[0]['items'][0]['itemid'])
```

```
    trigger = gettriggername(event[0]['triggers'][0]['triggerid'])
    clock = datetime.fromtimestamp(
        int(event[0]['clock'])).strftime(timeformat)
    value = triggervalues[int(event[0]['value'])]
    eventstable.append({"Host": host,
                        "Item": item,
                        "Trigger": trigger,
                        "Status": value,
                        "Time" : clock
                       })
```

Теперь, когда у нас есть вся информация о событиях, создадим выходной файл `.csv`:

```
filename = "events_" + appname + "_" + datetime.now().strftime(
    "%Y%m%d%H%M")
fieldnames = ['Host', 'Item', 'Trigger', 'Status', 'Time']
outfile = open(filename, 'w')
csvwriter = csv.DictWriter(outfile, delimiter=';',
                           fieldnames=fieldnames)
csvwriter.writerow(dict((h,h) for h in fieldnames))
for row in eventstable:
    csvwriter.writerow(row)
outfile.close()
```

Имя файла генерируется на основе имени приложения и времени составления отчета. Поскольку каждое событие в массиве `eventstable` является словарем, функции `csv.DictWriter` требуется передать массив `fieldnames`, чтобы определить порядок вывода полей. Далее, перед циклом вывода фактических данных из массива `eventstable`, выводится строка заголовка.

Сценарий выше можно расширить в разных направлениях, добавив в него вывод других полезных данных. Ниже перечислено несколько примеров возможных расширений, но вообще их спектр ограничивается только вашим воображением:

- предусмотреть передачу в сценарий периода времени для ограничения количества извлекаемых событий;
- предусмотреть упорядочение событий по хостам и триггерам;
- добавить вычисления для определения продолжительности событий на основе изменений состояний триггеров;
- выводить подтвержденные данные.

Создание графиков на основе данных

К настоящему моменту вам должны быть известны мощные возможности Zabbix по визуализации данных. В веб-интерфейсе можно создавать и отображать самые разные графики, карты и диаграммы, помогающие анализировать исторические данные, изменение состояний триггеров с течением времени, доступность служб и т. д. Подобно всем остальным инструментам Zabbix, все функции визуализации также доступны через прикладной интерфейс. Можно, например, написать про-

граммы создания, изменения и визуализации комплексных экранов, графиков и карт сетей, но что-то подобное вам едва ли придется делать, если только вы не соберетесь написать свой, альтернативный интерфейс к системе мониторинга.

С другой стороны, было бы интересно извлечь и визуализировать данные, которые сложно анализировать с применением стандартных возможностей веб-интерфейса из-за их большой рассеянности. Примером таких данных могут служить зависимости триггеров. Как рассказывалось в *главе 6 «Управление оповещениями»*, триггер может зависеть от одного или нескольких других триггеров и не переключаться в состояние `PROBLEM`, если один из триггеров, от которых он зависит, уже находится в этом состоянии.

Это была бы очень удобная функция, так как нет иного способа охватить одним взглядом все триггеры, от которых зависят другие триггеры, а также триггеры, от которых зависят те триггеры, и т. д. Решить эту задачу можно с помощью пакета Graphviz и пары строк кода на Python.

Пакет программ Graphviz

Graphviz (http://www.graphviz.org) – это набор программ для визуализации данных в графическом виде. С их помощью можно создавать графики произвольной сложности из специально сформированных текстовых данных. Пакет поддерживает множество средств визуализации данных и порой очень сложен в использовании, но вы легко сможете создать базовую функциональную основу и затем использовать ее для разработки своих сценариев.

Установите пакет Graphviz, если он еще не установлен в вашей системе. В Red Hat Enterprise Linux установку можно выполнить командой:

```
# yum install 'graphviz*'
```

Программа создания графов называется `dot`. Она принимает описание графа в текстовом виде и генерирует соответствующее изображение в выбранном формате. Ниже приводится пример описания графа:

```
digraph G {
  main → node1 → node2;
  main → node3;
  main → end;
  node2 → node4;
  node2 → node5;
  node3 → node4;
  node4 → end;
}
```

Поместите это описание в файл `graph.gv` и выполните следующую команду:

```
$ dot -Tpng graph.gv -o graph.png
```

В результате вы получите изображение в формате PNG, которое выглядит, как показано на рис. 9.4.

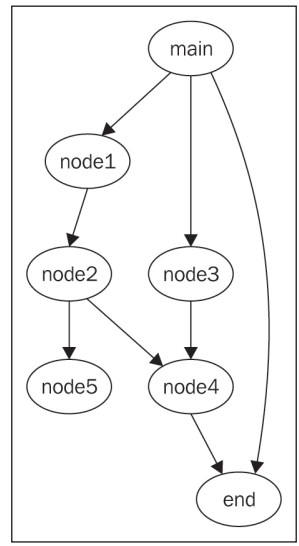


Рис. 9.4 ❖ Пример отображения графа

Как видите, с помощью пакета Graphviz легко можно нарисовать граф зависимостей между триггерами после извлечения необходимой информации с помощью API. А теперь посмотрим, как это реализовать на практике.

Создание графа зависимостей триггеров

Ниже приводится сценарий на языке Python, который извлекает данные о зависимостях между триггерами и выводит описание графа на языке программы `dot`:

```
#!/usr/bin/python
from pyzabbix import ZabbixAPI
zh = ZabbixAPI("https://127.0.0.1/zabbix")
zh.login(user="Admin", password="zabbix")

def gettriggername(triggerid=''):
    return zh.trigger.get(triggerids=triggerid,
        output=['description'])[0]['description']
```

В первой части сценария нет ничего нового. Здесь открывается сеанс работы с Zabbix API и определяется простая вспомогательная функция.

```
tr = zh.trigger.get(selectDependencies="refer",
                    expandData="1",
                    output="refer")
dependencies = [(t['dependencies'], t['host'], t['triggerid'])
                for t in tr if t['dependencies'] != [] ]
```

Следующие строки извлекают все триггеры с их зависимостями и создают список, попутно отбрасывая триггеры, не имеющие зависимостей.

```
outfile = open('trigdeps.gv', 'w')
outfile.write('digraph TrigDeps {\n')
outfile.write('graph[rankdir=LR]\n')
outfile.write('node[fontsize=10]\n')
```

Далее в выходной файл выводится несколько строк с настройками графа, которые определяют направление (слева направо) и размер шрифта для подписей узлов.

```
for (deps, triggerhost, triggerid) in dependencies:
    triggername = gettriggername(triggerid)

    for d in deps:
        depname = gettriggername(d['triggerid'])
        dephost = d['host']
        edge = '"{}:\\n{}" -> "{}:\\n{}";'.format(
            dephost, depname, triggerhost, triggername)
        outfile.write(edge + '\n')
```

Эти строки составляют основу сценария. Вложенный цикл `for` необходим, потому что единственный триггер может иметь несколько зависимостей, которые все должны быть отображены в графе. Для каждой найденной зависимости между триггерами в файле с описанием графа определяется ребро.

```
outfile.write('}\n')
outfile.close()
```

По достижении конца списка остается только закрыть описание графа и сам файл.

Попробуем выполнить сценарий:

```
$ chmod +x triggerdeps.py
$ ./triggerdeps.py
```

В результате получится файл `trigdeps.gv` с примерно таким содержимым:

```
digraph TrigDeps {
graph[rankdir=LR]
node[fontsize=10]
"Template IPMI Intel SR1630:\nPower" -> "Template IPMI Intel SR1630:\
nBaseboard Temp Critical [{ITEM.VALUE}]";
"Template IPMI Intel SR1630:\nBaseboard Temp Critical [{ITEM.VALUE}]"
-> "Template IPMI Intel SR1630:\nBaseboard Temp Non-Critical [{ITEM.
VALUE}]";
"Template IPMI Intel SR1630:\nPower" -> "Template IPMI Intel SR1630:\
nBaseboard Temp Non-Critical [{ITEM.VALUE}]";
"Template IPMI Intel SR1630:\nPower" -> "Template IPMI Intel SR1630:\
nBB +1.05V PCH Critical [{ITEM.VALUE}]";
"Template IPMI Intel SR1630:\nBB +1.05V PCH Critical [{ITEM.VALUE}]"
-> "Template IPMI Intel SR1630:\nBB +1.05V PCH Non-Critical [{ITEM.
VALUE}]";
```

```
"Template IPMI Intel SR1630:\nPower" -> "Template IPMI Intel SR1630:\
nBB +1.05V PCH Non-Critical [{ITEM.VALUE}]";
"Template IPMI Intel SR1630:\nPower" -> "Template IPMI Intel SR1630:\
nBB +1.1V P1 Vccp Critical [{ITEM.VALUE}]";
}
```

Если передать этот файл программе `dot`, она создаст файл изображения с графом:

```
$ dot -Tpng trigdeps.gv -o trigdeps.png
```

Диаграмма может получиться очень большой, как показано на рис. 9.5.

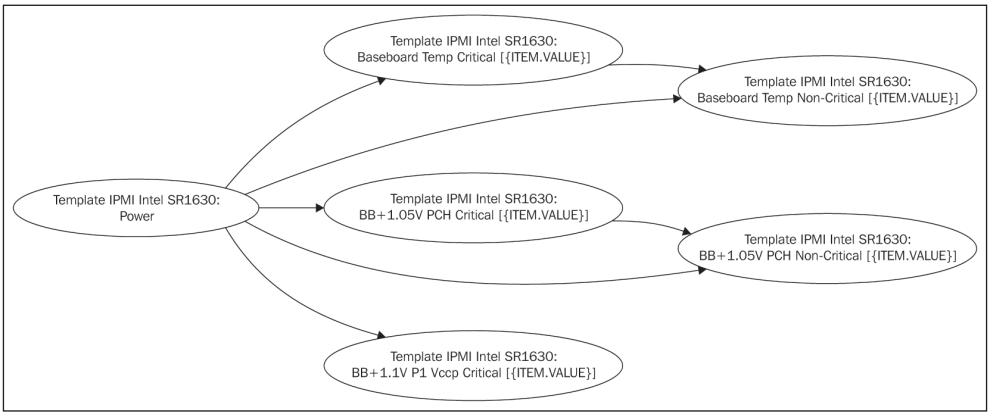


Рис. 9.5 ❖ Результат попытки получить граф зависимостей триггеров

Помимо формы и расположения узлов графа, а также интеграции функций формирования графов в сценарии на Python, существует большое количество других направлений для усовершенствования сценария. Например, с помощью методов API можно поместить полученное изображение на карту Zabbix или решить обратную задачу и установить зависимости между триггерами, опираясь на внешнее определение.

Создание карт Zabbix на основе файлов с описанием

Было бы интересно посмотреть, как, начиная с файла, включающего определение графа для программы `dot`, создать карту Zabbix автоматизированным способом. Проблема автоматизации представляет особый интерес, потому что Zabbix страдает некоторыми проблемами, например:

1) не поддерживается возможность одновременного перемещения нескольких элементов: https://support.zabbix.com/browse/ZBXNEXT-161;
2) не поддерживается массовое добавление хостов: https://support.zabbix.com/browse/ZBXNEXT-163;
3) не поддерживается клонирование существующих компонентов карты: https://support.zabbix.com/browse/ZBXNEXT-51;

4) отсутствует возможность автоматического выбора ярлыков, чтобы можно было проверить соответствие их размеров размерам карты: https://support.zabbix.com/browse/ZBXNEXT-1608.

Этого набора пунктов уже достаточно, чтобы задуматься об автоматизации для ускорения выполнения рутинных задач. Graphviz предоставляет в наше распоряжение отличный инструмент для создания изображения и преобразования его в вызовы Zabbix API. Для этого нам потребуется:

1) прочитать файл с описанием графа;
2) сгенерировать топологию с применением `graphviz`;
3) извлечь все координаты из сгенерированной топологии;
4) подключиться к серверу Zabbix с помощью PyZabbix;
5) сгенерировать топологию автоматическим способом.

Теперь, наконец, можно приступать к программированию на Python. Следующий код заимствован из примера, представленного Волькером Фройликом (Volker Fröhlich). Он был немного изменен, так как оригинал плохо работал с версией Zabbix 2.4.

В первых строках выполняется импортирование библиотек `ZabbixAPI` и `networkx`:

```
import networkx as nx
from pyzabbix import ZabbixAPI
```

Затем определяется файл в формате Graphviz DOT, который используется как источник данных. Здесь файл создается путем экспортирования данных из системы Zabbix, с учетом всех отношений между узлами. В данном примере используется простая строчка кода:

```
dot_file="/tmp/example.dot"
```

В следующих строках определяются имя пользователя, пароль, размеры карты и ее имя:

```
username="Admin"
password="zabbix"
width = 800
height = 600
mapname = "my_network"
```

Далее следует набор констант, определяющих типы элементов:

```
ELEMENT_TYPE_HOST = 0
ELEMENT_TYPE_MAP = 1
ELEMENT_TYPE_TRIGGER = 2
ELEMENT_TYPE_HOSTGROUP = 3
ELEMENT_TYPE_IMAGE = 4
ADVANCED_LABELS = 1
LABEL_TYPE_LABEL = 0
```

Затем определяются используемые ярлыки и цвета:

```
icons = {
    "router": 23,
```

```
    "cloud": 26,
    "desktop": 27,
    "laptop": 28,
    "server": 29,
    "sat": 30,
    "tux": 31,
    "default": 40,
}

colors = {
    "purple": "FF00FF",
    "green": "00FF00",
    "default": "00FF00",
}
```

Теперь определим несколько функций: первая управляет регистрацией, а вторая выполняет поиск хоста:

```
def api_connect():
    zapi = ZabbixAPI("http://127.0.0.1/zabbix/")
    zapi.login(username, password)
    return zapi

def host_lookup(hostname):
    hostid = zapi.host.get({"filter": {"host": hostname}})
    if hostid:
        return str(hostid[0]['hostid'])
```

После этого сценарий читает файл в формате dot и преобразует его в граф:

```
G=nx.read_dot(dot_file)
```

Затем он открывает граф:

```
pos = nx.graphviz_layout(G)
```

Здесь есть возможность выбрать предпочтительный алгоритм. Graphviz поддерживает несколько видов компоновки, с помощью которых можно менять оформление карты. За дополнительной информацией обращайтесь к официальной документации, доступной по адресу: http://www.graphviz.org/.

Следующий шаг после создания графа – поиск максимальных координат компоновки. Это даст возможность выполнить масштабирование карты:

```
positionlist=list(pos.values())
maxpos=map(max, zip(*positionlist))
for host, coordinates in pos.iteritems():
    pos[host] = [int(coordinates[0] * width/maxpos[0] * 0.95 -
                     coordinates[0] * 0.1),
                 int((height - coordinates[1] * height/maxpos[1]) *
                     0.95 + coordinates[1] * 0.1)]
nx.set_node_attributes(G,'coordinates',pos)
```

Graphviz и Zabbix используют разные системы координат – в Graphviz начало координат находится в левом нижнем углу, а в Zabbix – в левом верхнем.

Далее нужно извлечь список элементов selementids, который понадобится для определения связей между узлами:

```
selementids = dict(enumerate(G.nodes_iter(), start=1))
selementids = dict((v,k) for k,v in selementids.iteritems())
nx.set_node_attributes(G,'selementid',selementids)
nx.set_node_attributes(G,'selementid',selementids)
```

Теперь определим карту в Zabbix, ее имя и размеры:

```
map_params = {
    "name": mapname,
    "label_type": 0,
    "width": width,
    "height": height
}
element_params=[]
link_params=[]
```

Наконец, установим соединение с сервером Zabbix:

```
zapi = api_connect()
```

Подготовим всю информацию об узлах и координатах и затем установим ярлыки:

```
for node, data in G.nodes_iter(data=True):
    # общая часть
    map_element = {}
    map_element.update({
        "selementid": data['selementid'],
        "x": data['coordinates'][0],
        "y": data['coordinates'][1],
        "use_iconmap": 0,
    })
```

Проверим наличие имени хоста:

```
    if "hostname" in data:
        map_element.update({
            "elementtype": ELEMENT_TYPE_HOST,
            "elementid": host_lookup(
                data['hostname'].strip('"')),
            "iconid_off": icons['server'],
        })
    else:
        map_element.update({
            "elementtype": ELEMENT_TYPE_IMAGE,
            "elementid": 0,
        })
```

Добавим подписи к изображениям:

```
    if "label" in data:
        map_element.update({
            "label": data['label'].strip('"')
        })
    if "zbximage" in data:
        map_element.update({
            "iconid_off": icons[data['zbximage'].strip('"')],
        })
    elif "hostname" not in data and "zbximage" not in data:
        map_element.update({
            "iconid_off": icons['default'],
        })

    element_params.append(map_element)
```

Теперь нужно обойти все ребра, чтобы создать связи между элементами, хранящимися в списке selementids:

```
nodenum = nx.get_node_attributes(G,'selementid')
for nodea, nodeb, data in G.edges_iter(data=True):
    link = {}
    link.update({
        "selementid1": nodenum[nodea],
        "selementid2": nerodenum[nodeb],
    })

    if "color" in data:
        color = colors[data['color'].strip('"')]
        link.update({
            "color": color
        })
    else:
        link.update({
            "color": colors['default']
        })

    if "label" in data:
        label = data['label'].strip('"')
        link.update({
            "label": label,
        })

    link_params.append(link)

# включить в карту поготовленную информацию
map_params["selements"] = element_params
map_params["links"] = link_params
```

После заполнения всех параметров в map_params их нужно передать Zabbix API:

```
map=zapi.map.create(map_params)
```

Программа закончена, и мы можем опробовать ее! В случае из практики для заполнения топологии с 2500 хостами потребовалось всего 2–3 секунды!

Здесь мы проверим работу программы с исходным файлом описания графа, содержащим 24 хоста:

```
[root@localhost]# time ./Generate_MyMap.py
real    0m0.005s
user    0m0.002s
sys     0m0.003s
```

Как видите, программа действительно работает очень быстро.., но давайте посмотрим, что получилось в результате. На рис. 9.6 изображена карта, сгенерированная автоматически за 0,005 секунды:

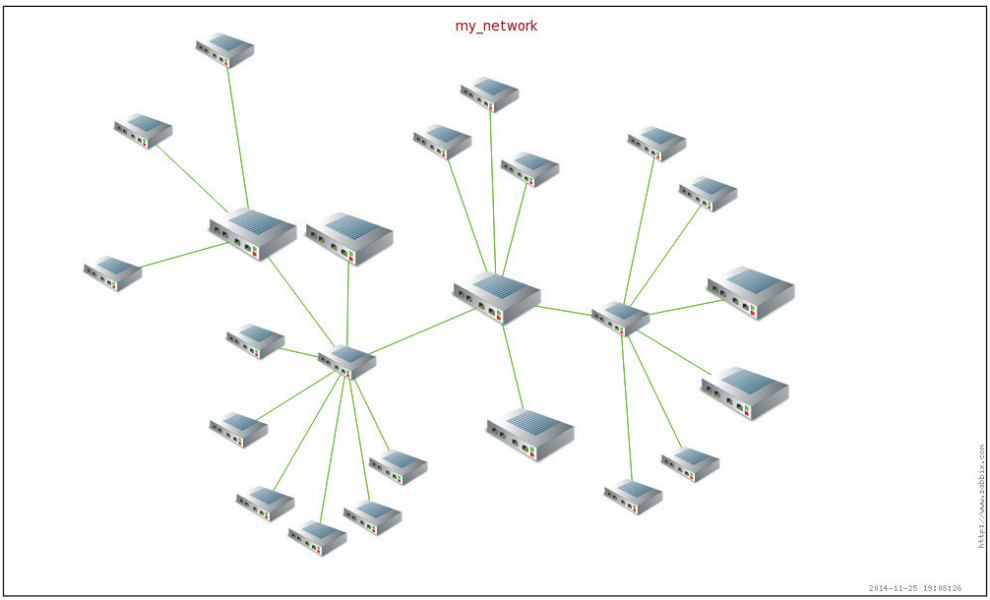


Рис. 9.6 ❖ Автоматически сгенерированная карта с 24 хостами

Цель данного примера состояла в том, чтобы показать, насколько просто автоматизировать сложные и обременительные задачи с использованием Zabbix API. Тот же метод можно использовать для выполнения подготовительных операций во время запуска. Кроме того, существует множество современных инструментов, способных предоставить данные об уже контролируемых хостах, например Cisco Prime, или другие специализированные инструменты, с помощью которых можно извлекать значительные объемы данных, преобразовывать их в формат dot и заполнять карту хостов в Zabbix.

В заключение

В этой главе мы только коснулись имеющихся возможностей Zabbix API. Если в процессе чтения главы прорабатывали примеры, значит, вы уже достаточно уверенно владеете протоколом JSON-RPC, который составляет основу API. Теперь вы знаете, как исследовать различные методы, и имеете представление о том, как использовать их для управления системой Zabbix или расширения возможностей обработки данных.

Этим обсуждением API мы завершаем исследование возможностей Zabbix. В следующей, заключительной главе мы используем знания, полученные к данному моменту, для более плотной интеграции Zabbix в вашу информационно-вычислительную инфраструктуру, наладив взаимодействие с другими системами управления.

Глава 10

Интеграция с Zabbix

Система мониторинга по определению требует взаимодействий с другими системами. С одной стороны, она должна подключаться к контролируемым объектам, чтобы получить измеряемые параметры и определить состояние службы. С другой – она должна поставлять собранную информацию вовне, чтобы системные администраторы могли проанализировать данные и оповещения. В предыдущих главах основное внимание уделялось первой части уравнения, а именно сбору данных, и всегда предполагали, что вторая часть – экспорт данных и оповещений – сопряжена с отправкой последовательности сообщений обслуживающему персоналу. Это верно для большинства конфигураций, и такая организация составляет основу любой инфраструктуры Zabbix, но так же верно и то, что она может оказаться весьма ограниченной в больших и сложных окружениях.

Любая система управления имеет свое характерное представление о своем окружении, которое напрямую зависит от выполняемых ею функций. Системы управления идентификацией хранят всю информацию о пользователях, паролях и привилегиях, тогда как система инвентаризации хранит сведения об аппаратных и программных конфигурациях. Системы управления жалобами клиентов хранят сведения о текущих проблемах, возникающих у пользователей, а системы мониторинга определяют состояние доступности и параметры производительности всего, к чему они могут подключиться. Так как многие из этих систем совместно используют некоторые общие данные, например информацию о пользователях, параметры подключения и т. д., очень важно, чтобы эта информация свободно передавалась между системами без ручного вмешательства администраторов.

Совершенно понятно, что ни Zabbix, ни какая другая система мониторинга не способна поддерживать интеграцию с любыми другими системами, имеющимися в мире. Но открытая природа, простой протокол и мощный прикладной программный интерфейс (API) способны существенно упростить интеграцию системы мониторинга с любыми другими инструментами управления, развернутыми в вашем окружении. Тема интеграции настолько обширна, что заслуживает отдельной книги, тем не менее мы попытаемся начать ее обсуждение на примере Zabbix.

В этой главе вы увидите конкретный пример интеграции Zabbix с WhatsApp™ и Zabbix с **Request Tracker (RT)**. Я думаю, не нужно пояснять, что WhatsApp – это известная система обмена сообщениями, которая поддерживает шифрование

и даже способна выполнять звонки по телефону с использованием VoIP, а Request Tracker – это открытая система управления проблемами, выпускаемая компанией Best Practical (http://bestpractical.com/rt/).

К концу этой главы вы научитесь:

- посылать оповещения своим системным администраторам через WhatsApp;
- интегрировать в Zabbix собственные средства рассылки сообщений;
- пересылать предупреждения в систему управления проблемами;
- следить за соответствием проблем и событий;
- квитировать события, определяя статус проблем;
- автоматически закрывать проблемы по возвращении триггеров в состояние `OK`.

Здесь мы не будем обсуждать никаких новых понятий или функций Zabbix, а просто обсудим приемы использования существующих приложений с применением средств, которые уже изучили в предыдущих главах.

Интеграция с WhatsApp

Система WhatsApp получила настолько широкую известность, что не требует представления. С другой стороны, интересно отметить, что было разработано большое количество библиотек для работы с протоколом WhatsApp. В этом примере для взаимодействий с WhatsApp мы будем использовать библиотеку Yowsup на языке Python. В течение года я опробовал большое количество библиотек, разработанных для поддержки этой службы, и остановил свой выбор на Yowsup, которая применялась в проекте Wazapp для создания неофициального клиента WhatsApp для Nokia N9, использующегося более чем 200 000 пользователями. Также на основе библиотеки Yowsup был создан клиент для Blackberry 10 – надежный компонент, который вы можете использовать для интеграции своей системы мониторинга.

Давайте для начала определим требования:

- Python 2.6+ или Python 3.0+;
- пакеты для Python: `python-dateutil`;
- пакеты поддержки шифрования для Python: `protobuf`, `pycrypto` и `python-axolotl-curve25519`;
- пакеты поддержки `yowsup-cli` для Python: `argparse`, `readline` и `pillow` (для отправки изображений).

Теперь можно установить перечисленные пакеты с помощью `yum`:

```
$ yum install python python-dateutil python-argparse
```

Как обычно, диспетчер пакетов `yum` автоматически определит и удовлетворит все зависимости. После установки пакетов загрузим библиотеку Yowsup. В этом случае вам решать, что лучше: скопировать репозиторий Git или загрузить пакет `master` с сайта проекта. В этом примере я предлагаю загрузить пакет:

```
# wget https://github.com/tgalal/yowsup/archive/master.zip
```

После загрузки распакуйте архив:

```
# unzip master.zip
```

В результате будет создано дерево каталогов, повторяющее структуру репозитория Git. Теперь перейдите в корневой каталог распакованного архива:

```
# cd ./yowsup-master
```

И запустите сборку проекта. Для этого необходимо, чтобы в системе имелся пакет `python-devel`, который можно установить командой

```
# yum install -y python-devel
```

Итак, запустите сборку проекта:

```
# python setup.py install
```

Сценарий `setup.py` автоматически выявит и установит все зависимости, что избавит вас от необходимости устанавливать их вручную с помощью диспетчера `pip`.

Подготовка к отправке сообщений

Теперь, после установки пакета, необходимо сначала создать конфигурационный файл, как показано ниже:

```
# cat ./yowsup.config
cc=
phone=
id=
password=
```

В поле `cc` следует указать код страны.

Поле `phone` должно включать: *код страны* + *код области* + *номер телефона*. Имейте в виду, что код страны не должен включать начальный символ «+» или «00».

Поле `id` используется для регистрации звонков и журналирования. Недавно, в обновленных версиях клиентов WhatsApp, использование IMEI/MAC для создания пароля учетной записи было отключено. Но сама функция все еще поддерживается, если указать ключ `--v1`. Обычно это поле должно содержать код IMEI телефона, если учетная запись настраивалась в устройстве Nokia или на Android, или MAC-адрес устройств на iOS. Если вы не собираетесь использовать имеющиеся учетные данные, оставьте это поле пустым или вообще удалите его.

Наконец, поле `password` должно содержать пароль учетной записи. Вы получите этот пароль, когда зарегистрируете номер телефона с использованием `yowsup-cli`. Если ваш номер еще не зарегистрирован, оставьте это поле пустым и заполните его, когда пройдете процедуру регистрации.

Я рекомендую установить права доступа к конфигурационному файлу `600`, и, поскольку доступ к нему будет осуществляться из учетной записи сервера Zabbix,

вы сможете настроить роль `sudo` для нее и тем самым обеспечить неплохой уровень защищенности. После этого только сервер Zabbix сможет посылать сообщения.

Регистрация клиента yowsup

Теперь пришло время зарегистрировать клиента и тем самым дать ему возможность посылать сообщения.

Прежде всего нам потребуется номер телефона, который будет использоваться этим приложением. Важно для этой цели использовать действительный номер телефона, на который можно принимать сообщения SMS.

Для регистрации клиента необходим конфигурационный файл `yowsup.config`, описанный выше, с заполненными полями `id` и `phone`. Остальные поля пока можно оставить пустыми. Собственно регистрация выполняется командой:

```
# ./yowsup-cli registration -c ./yowsup.config -r sms

INFO:yowsup.common.http.warequest:{"status":"sent","length":6,"method":"sms","retry_
after":1805}

status: sent

retry_after: 1805

length: 6

method: sms

#
```

По завершении на указанный номер телефона должно прийти сообщение SMS с кодом в форме *NNN-NNN*. Получив код, введите команду:

```
# ./yowsup-cli registration -c ./yowsup.config -R 117-741

INFO:yowsup.common.http.warequest:{"status":"ok","login":"41076XXXXXX","pw":"w3cp6Vb7UAU
lKG6/xhx/1K4hA=","type":"existing","expiration":1465119599,"kind":"free","price":"\u20ac
0,89","cost":"0.89","currency":"EUR","price_expiration":1439763526}

status: ok

kind: free

pw: w3cp6Vb7UAUlKG6/xhx/1K4hA=

price: € 0,89

price_expiration: 1439763526

currency: EUR

cost: 0.89

expiration: 1465119599

login: 41076XXXXXX
```

```
type: existing

#
```

Теперь у нас имеется пароль в формате BASE64, в предыдущем листинге он определен в строке `pw: w3cp6Vb7UAUlKG6/xhx/1K4hA=`. Добавьте его в конфигурационный файл `yowsup.config`.

Отправка первого сообщения в WhatsApp

Наконец, у нас все готово к работе. Давайте сначала попробуем отправить какое-нибудь сообщение. Далее во всех тестах мы будем использовать новую учетную запись `yousup`:

```
# $HOME/yowsup-master/yowsup-cli demos -c ./yowsup.config -s 4176XXXXX "Test message form cli"

WARNING:yowsup.stacks.yowstack:Implicit declaration of parallel layers in a tuple is
deprecated, pass a YowParallelLayer instead

INFO:yowsup.demos.sendclient.layer:Message sent

Yowsdown
```

Отправьте еще одно сообщение и проверьте его доставку, например:

```
# $HOME/yowsup-master/yowsup-cli demos -c ./yowsup.config -s 4176XXXXX "Test message form
cli. 2nd test"

WARNING:yowsup.stacks.yowstack:Implicit declaration of parallel layers in a tuple is
deprecated, pass a YowParallelLayer instead

INFO:yowsup.demos.sendclient.layer:Message sent
```

Результаты можно наблюдать на своем телефоне или в веб-приложении WhatsApp, как показано на рис. 10.1.

Теперь рассмотрим особенности использования.

Настройка безопасности клиента yowsup

Прежде чем двигаться дальше, имеет смысл ограничить доступ к клиенту `yowsup`, разрешив использовать его только учетной записи Zabbix.

Для этого создайте специальную учетную запись, например `yowsup`. Затем от имени root выполните следующую команду:

```
# useradd yowsup
```

Определите пароль для этой учетной записи:

```
# passwd yowsup
Changing password for user yowsup.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

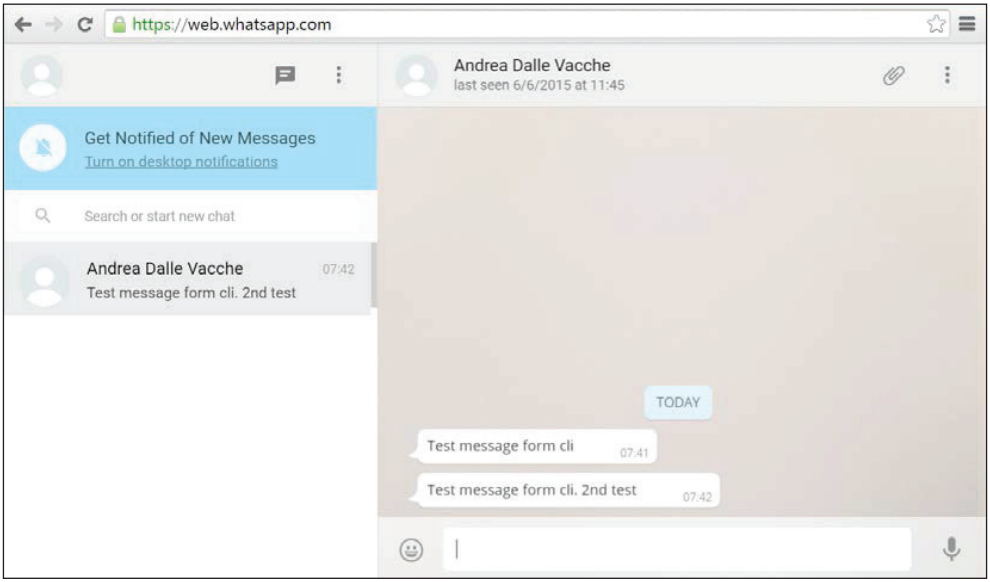


Рис. 10.1 ❖ Результаты отправки тестовых сообщений

Отредактируйте файл `sudoers`, чтобы дать право учетной записи сервера Zabbix выполнять необходимые команды:

```
#visudo -f /etc/sudoers.d/Zabbix
```

Добавьте в файл следующие строки:

```
zabbix ALL=(ALL) NOPASSWD: /usr/bin/sudo -l
zabbix ALL=(ALL) NOPASSWD: /home/yowsup/yowsup-master/yowsup-cli *
```

Теперь можно проверить возможность выполнять необходимые команды от имени пользователя Zabbix. Для этого введите следующую команду:

```
$ sudo -l
```

Вывод должен содержать следующие строки:

```
User zabbix may run the following commands on this host:
    (ALL) NOPASSWD: /usr/bin/sudo -l
    (ALL) NOPASSWD: /home/yowsup/yowsup-master/yowsup-cli *
```

В заключение скопируйте все файлы и данные в домашний каталог пользователя `yowsup`, выполнив следующие команды от имени root:

```
# cp -r -a yowsup-master /home/yowsup/
# chown -R yowsup:yowsup /home/yowsup/*
```

Библиотека Yowsup сохраняет всю историю операций в каталоге `$HOME/.yowsup/`, то есть команды выше копируют подготовленные перед этим файлы с настройками, что очень удобно. Не забывайте об этом.

Проверьте работоспособность еще раз, выполнив следующие команды от имени пользователя Zabbix:

```
$ sudo -u yowsup /home/yowsup/yowsup-master/yowsup-cli
Available commands:
==================
demos, version, registration
```

Если вы получили иной результат, проверьте файлы с настройками. Как заключительный тест попробуйте послать сообщение от имени Zabbix:

```
$ sudo -u yowsup /home/yowsup/yowsup-master/yowsup-cli demos -c /home/yowsup/yowsup-master/
yowsup.config -s 4176XXXXXX "Test message form zabbix 1st test"

WARNING:yowsup.stacks.yowstack:Implicit declaration of parallel layers in a tuple is
deprecated, pass a YowParallelLayer instead

INFO:yowsup.demos.sendclient.layer:Message sent

Yowsdown
```

Если все работает, как ожидается, вы должны увидеть сообщение в окне терминала и на странице веб-приложения WhatsApp (рис. 10.2).

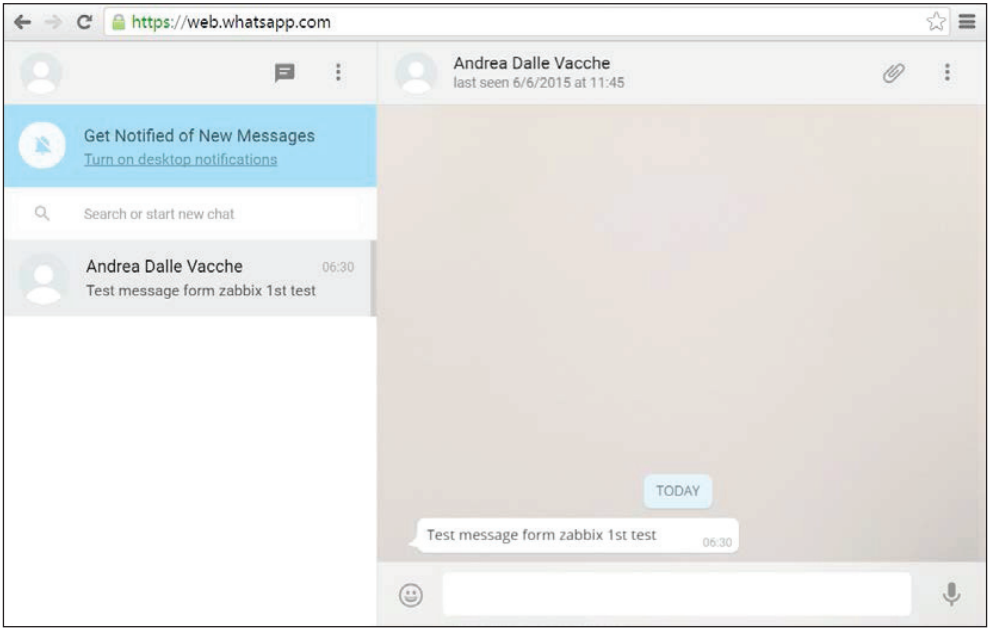


Рис. 10.2 ❖ Тестовое сообщение получено

Как видно на рис. 10.2, я благополучно получил сообщение, отправленное от имени Zabbix.

Создание первой группы в Zabbix для рассылки оповещений

Теперь, когда мы обезопасили конфигурацию рассылки оповещений, ограничив доступ к ней и наделив пользователя Zabbix необходимыми привилегиями, можно подумать о реализации фактического сценария оповещения. Мы уже проверили основные функции задействованного программного обеспечения, но в фактическом сценарии мы должны предусмотреть рассылку сообщений ограниченному кругу или группе лиц в соответствии с графиком смен, выходных дней, отпусков и других обстоятельств. С этой целью достаточно просто создать группу в WhatsApp. К счастью, программные компоненты поддерживают такую возможность и позволяют добавлять членов в группу или исключать их из нее.

Ниже демонстрируется, как создать группу с именем `zabbix_alert` для данного примера. Зарегистрируйтесь с учетной записью `yowsup` и выполните следующую команду:

```
# cd yowsup-master && ./yowsup-cli demos -c yowsup.config --yowsup
```

Она запустит клиента командной строки Yowsup. Фактически это интерактивная оболочка, позволяющая посылать команды приложению WhatsApp. После запуска клиент выводит следующее приветственное сообщение:

```
Yowsup Cli client
==================
Type /help for available commands
```

Если теперь ввести `/help`, клиент перечислит все поддерживаемые им функции; давайте попробуем:

```
[offline]:/help
-----------------------------------------------
/profile  setPicture  [path]                Set profile picture
/profile  setStatus   [text]                Set status text
/account  delete                             Delete your account
/group    info        [group_jid]            Get group info
/group    picture     [group_jid] [path]     Set group picture
/group    invite      [group_jid] [jids]     Invite to group. Jids are a comma separated
list
/group    leave       [group_jid]            Leave a group you belong to
/group    setSubject  [group_jid] [subject]  Change group subject
/group    demote      [group_jid] [jids]     Remove admin of a group. Jids are a comma
separated list
/group    promote     [group_jid] [jids]     Promote admin of a group. Jids are a comma
separated list
/group    kick        [group_jid] [jids]     Kick from group. Jids are a comma separated
list
/help                                        Print this message
/seq                                         Send init seq
```

```
/contacts sync        [contacts]           Sync contacts, contacts should be comma
separated phone numbers, with no spaces
/keys     set                              Send prekeys
/keys     get         [jids]               Get shared keys
/image    send        [number] [path]      Send and image
/presence available                        Set presence as available
/presence subscribe   [contact]            Subscribe to contact's presence updates
/presence unsubscribe [contact]            Unsubscribe from contact's presence updates
/presence name        [name]               Set presence name
/presence unavailable                      Set presence as unavailable
/ping                                      Ping server
/L                                         Quick login
/state    paused      [jid]                Send paused state
/state    typing      [jid]                Send typing state
/contact  picture     [jid]                Get profile picture for contact
/contact  picturePreview [jid]             Get profile picture preview for contact
/contact  lastseen    [jid]                Get lastseen for contact
/groups   create      [subject] [jids]     Create a new group with the specified subject
and participants. Jids are a comma separated list. Use '-' to keep group without
participants but you.
/groups   list                             List all groups you belong to
/disconnect Disconnect
/login                [username] [b64password] Login to WhatsApp
/ib       clean       [dirtyType]          Send clean dirty
/message  broadcast   [numbers] [content]  Broadcast message. numbers should comma
separated phone numbers
/message  send        [number] [content]   Send message to a friend
-----------------------------------------------
[offline]:
```

Одного взгляда достаточно, чтобы понять, что мы имеем дело с полноценным клиентом, поддерживающим все возможные операции, какие только могут понадобиться при работе со службой рассылки сообщений.

Прежде чем создать группу, необходимо выполнить вход. Обратите внимание, что оболочка сообщает вам ваш статус в строке приглашения к вводу. В данном случае мы все еще не подключены: `[offline]`. Благодаря тому что в команде запуска клиента, в параметре `-c`, указано имя конфигурационного файла, можно воспользоваться функцией быстрого входа:

```
[offline]:/L
Auth: Logged in!
[connected]:
```

Как видите, статус изменился на `[connected]`, и теперь можно посылать команды. Сейчас мы создадим группу командой `/groups create`, сопроводив ее именем группы и списком номеров телефонов ее членов. В данном примере указан только

один номер телефона, но вообще можно одной командой добавить все требуемые номера, перечислив их через запятую:

```
[connected]:/groups create zabbix_alert 4176XXXXXX
```

В результате эта команда выведет строки:

```
[connected]:INFO:yowsup.layers.protocol_groups.layer:Group create success
Iq:
ID: 1
Type: result
from: g.us

Notification: Notification
From: 39340XXXXXXX-1436940409@g.us
Type: w:gp2
Participant: 39340XXXXXXX@s.whatsapp.net
Creator: 39340XXXXXXX @s.whatsapp.net
Create type: new
Creation timestamp: 1436940409
Subject: zabbix_alert
Subject owner: 39340XXXXXXX@s.whatsapp.net
Subject timestamp: 1436940409
Participants: {39340XXXXXXX@s.whatsapp.net': 'admin', '4176XXXXXXX@s.whatsapp.net': None}
[connected]:
```

На рис. 10.3 показан результат выполнения этой команды в веб-приложении.

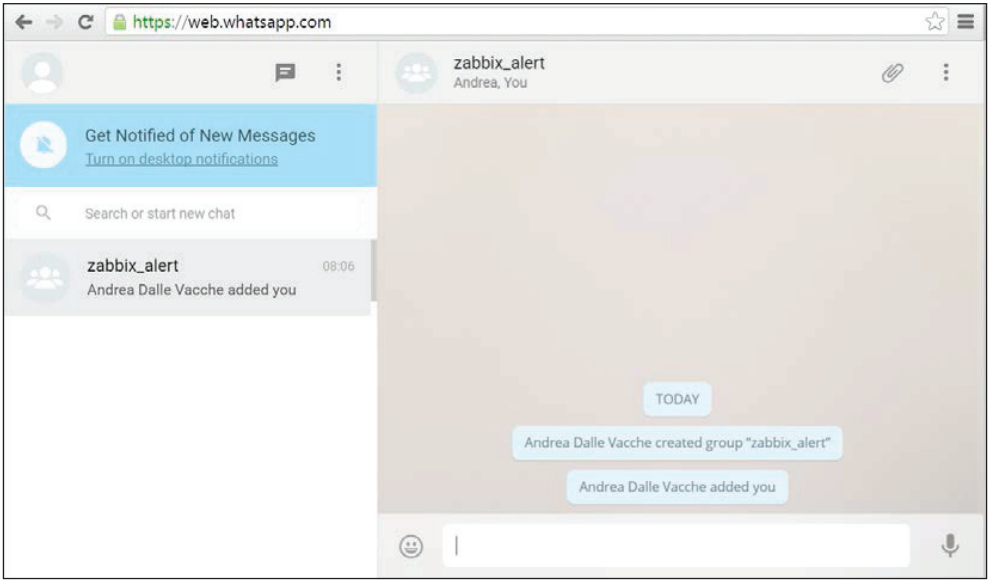


Рис. 10.3 ❖ Результат попытки добавить номер телефона в состав группы

Здесь группа получила идентификатор JID[1]:

```
From: 39340XXXXXXX-1436940409@g.us
```

Идентификатор JID состоит из номера телефона, создавшего группу, за которым следует уникальный идентификатор. Теперь все готово к отправке первого сообщения в группу. Попробуем сделать это, выполнив команду

```
# ./yowsup-cli demos -c ./yowsup.config -s 39340XXXXXXX-1436940409@g.us "Test message to
zabbix_alert group"
```

Ее результат показан на рис. 10.4.

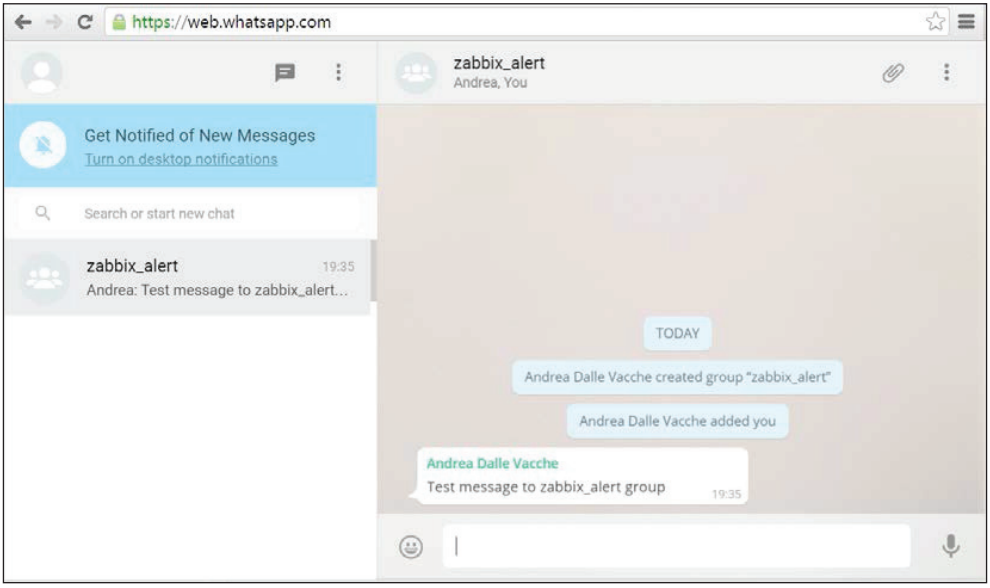


Рис. 10.4 ❖ Результат отправки сообщения группе

В качестве заключительного шага добавим еще одного администратора группы, так как гораздо надежнее, когда есть еще кто-то, кто сможет принимать решения в критических ситуациях, добавлять новых членов группы и т. д.

Для этого нужно запустить клиента и выполнить вход:

```
# ./yowsup-cli demos -c ./yowsup.config --yowsup
Yowsup Cli client
==================
Type /help for available commands
```

[1] Читается как «джид» – это аббревиатура от «Jabber Identifier (идентификатор Jabber). Так называются учетные записи в Jabber. Записывается JID по тому же принципу, что и адрес электронной почты. – *Прим. перев.*

```
[offline]:/L
Auth: Logged in!
[connected]:
```

Теперь выполним команду `/group`, указав идентификатор JID группы и список номеров, которые должны быть наделены полномочиями администратора. В данном примере добавим только один номер:

```
[connected]:/group promote 39340XXXXXXX-1436940409@g.us 4176XXXXXX
[connected]:INFO:yowsup.layers.protocol_groups.layer:Group promote participants success

[connected]:
```

Результат показан на рис. 10.5.

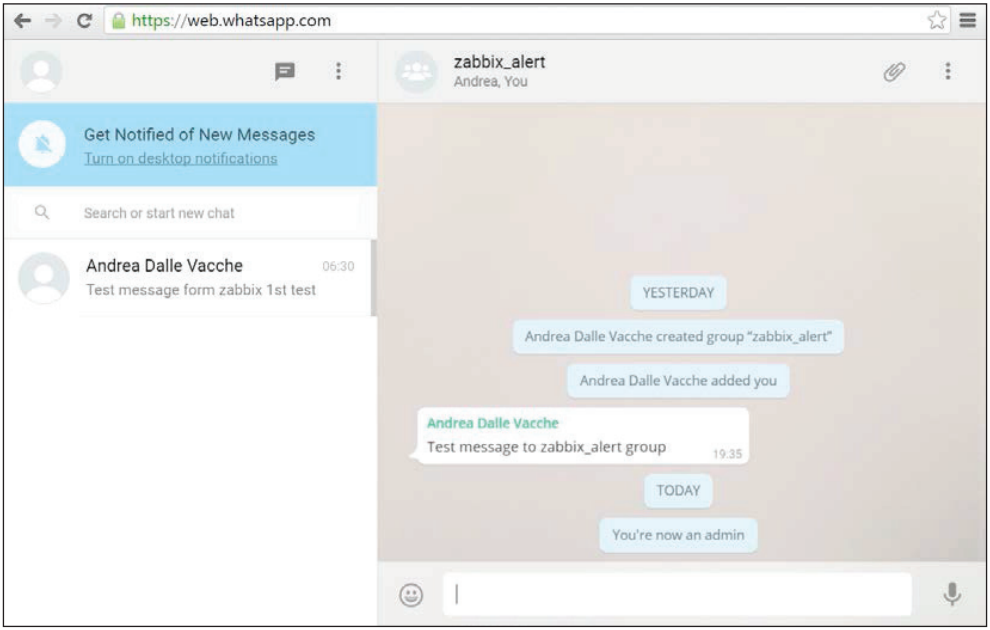


Рис. 10.5 ❖ Результат добавления еще одного администратора группы

Теперь я сам смогу добавлять контакты в группу и исключать их из нее.

Интеграция yowsup с Zabbix

Теперь все готово для интеграции Zabbix с WhatsApp. Для этого нужно написать сценарий, вызывающий утилиты командной строки с помощью команды `sudo`. Этот сценарий следует поместить в каталог, указанный в параметре `AlertScriptsPath`:

```
grep AlertScript /etc/zabbix/zabbix_server.conf

### Option: AlertScriptsPath
```

```
# AlertScriptsPath=${datadir}/zabbix/alertscripts
AlertScriptsPath=/usr/lib/zabbix/alertscripts
```

Итак, создайте сценарий `whatsapp.sh` в каталоге `/usr/lib/zabbix/alertscripts` со следующим содержимым:

```
$ cat /usr/lib/zabbix/alertscripts/whatsapp.sh
#!/bin/bash
BASEDIR=/home/yowsup/yowsup-master
sudo -u yowsup $BASEDIR/yowsup-cli demos -c $BASEDIR/yowsup.config -s $1 "$2 $3"
```

После этого необходимо определить новый метод оповещения, который будет использовать данный сценарий. Для этого перейдите на вкладку **Administration ⇨ Media type ⇨ Create media type** (Администрирование ⇨ Способы оповещений ⇨ Создать способ оповещения) и заполните форму, как показано на рис. 10.6.

Рис. 10.6 ❖ Форма с настройками нового типа оповещения

Далее требуется определить действие, использующее новый способ оповещения. Для этого перейдите на вкладку **Configuration ⇨ Actions** (Настройка ⇨ Действия), выберите в раскрывающемся меню пункт **Trigger** (Триггер) и щелкните на кнопке **Create action** (Создать действие), как показано на рис. 10.7.

Затем на вкладке **Operations** (Операции) определите, кому должно посылаться данное сообщение. В данном примере решено, что сообщение будет посылаться всем членам группы администраторов Zabbix (**Zabbix administrators**), как показано на рис. 10.8.

Теперь следует настроить метод оповещения в учетных записях пользователей (в данном случае членов группы **Zabbix administrators**). Для этого перейдите на вкладку **Administration ⇨ Users** (Администрирование ⇨ Пользователи), выберите пользователя и укажите тип оповещения **whatsapp**. Затем введите номер телефона без префиксов «+» и «00» перед кодом страны, как показано на рис. 10.9.

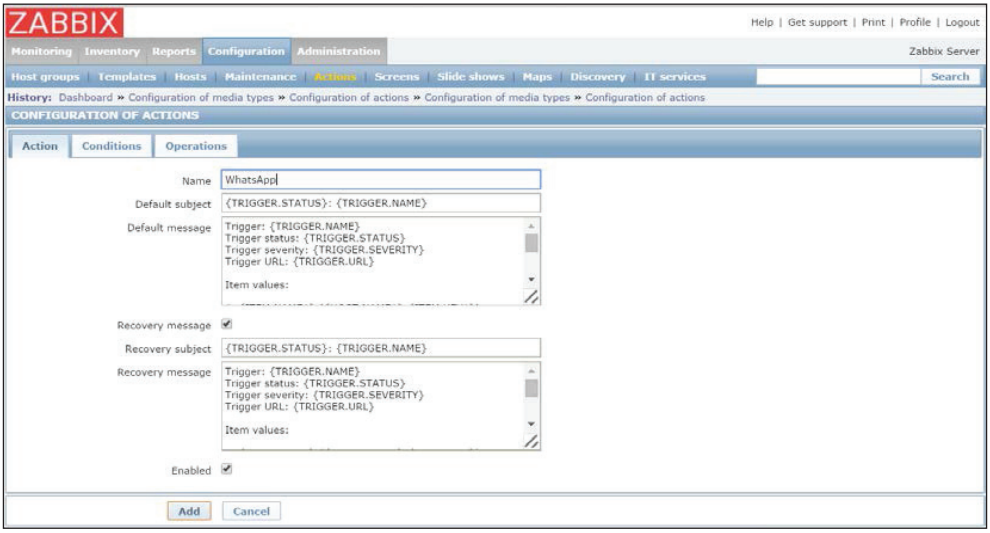


Рис. 10.7 ❖ Создание действия, использующего новый способ оповещения

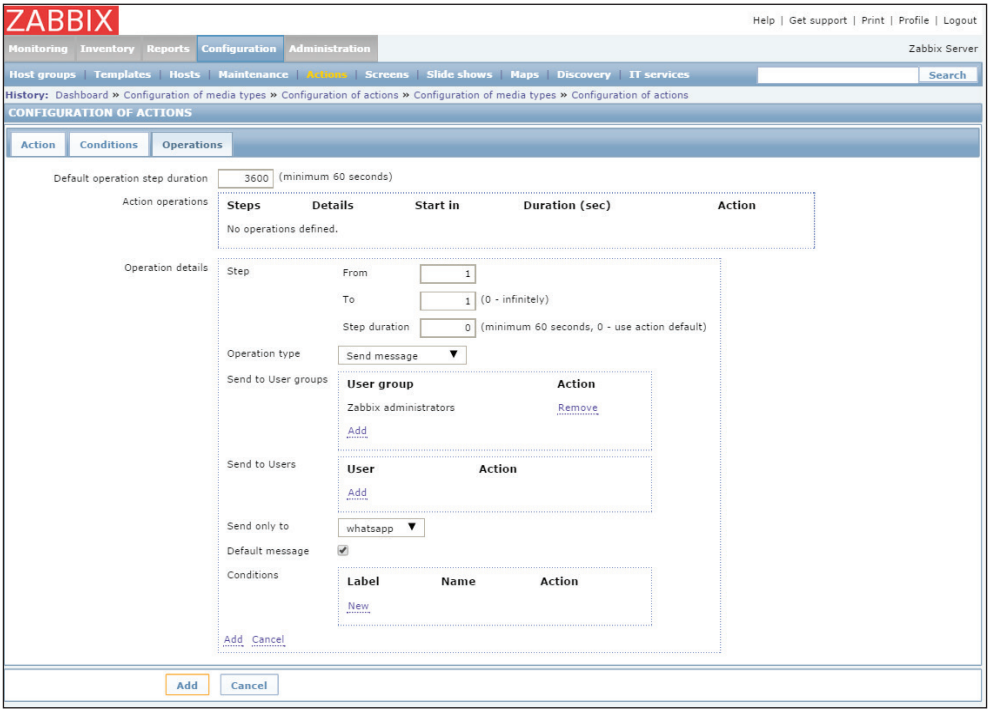


Рис. 10.8 ❖ Определение получателей сообщения

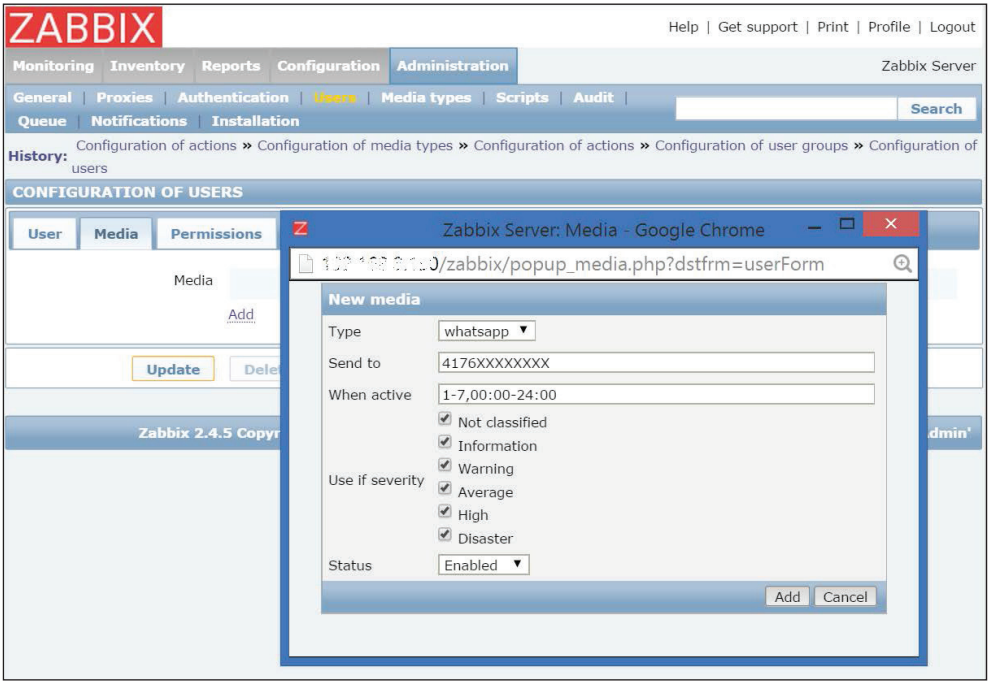


Рис. 10.9 ❖ Настройка учетной записи пользователя для получения сообщений

Здесь также можно выбрать уровни важности сообщений для данного способа оповещения.

В данном случае можно пойти двумя путями – либо настроить рассылку сообщений всем пользователям, перечисленным в разделе **Media** (Оповещения), либо использовать группу WhatsApp. В данном примере можно просто определить группу `39340XXXXXXX-14369404090g.us` с единственной учетной записью (созданную выше).

Наконец, можно провести тестирование выполненных настроек и понаблюдать за потоком сообщений, перейдя на вкладку **Administration** ⇨ **Audit** (Администрирование ⇨ Аудит) и выбрав фильтр **Action log** (Журналы). На этой вкладке можно видеть все выполняемые действия. На рис. 10.10 можно видеть событие, которое я инициировал, чтобы проверить работу всей связки. В данном случае событие обусловлено добавлением временного правила в `iptables`.

Я также немного изменил сценарий `whatsapp.sh`, чтобы проследить за его работой:

```
$ cat /usr/lib/zabbix/alertscripts/whatsapp.sh
#!/bin/bash
BASEDIR=/home/yowsup/yowsup-master
echo "sudo -u yowsup $BASEDIR/yowsup-cli demos -c $BASEDIR/yowsup.config -s $1 \"$2 $3\""
```

```
>> /var/log/whatsapp.log
sudo -u yowsup $BASEDIR/yowsup-cli demos -c $BASEDIR/yowsup.config -s $1 "$2 $3"
```

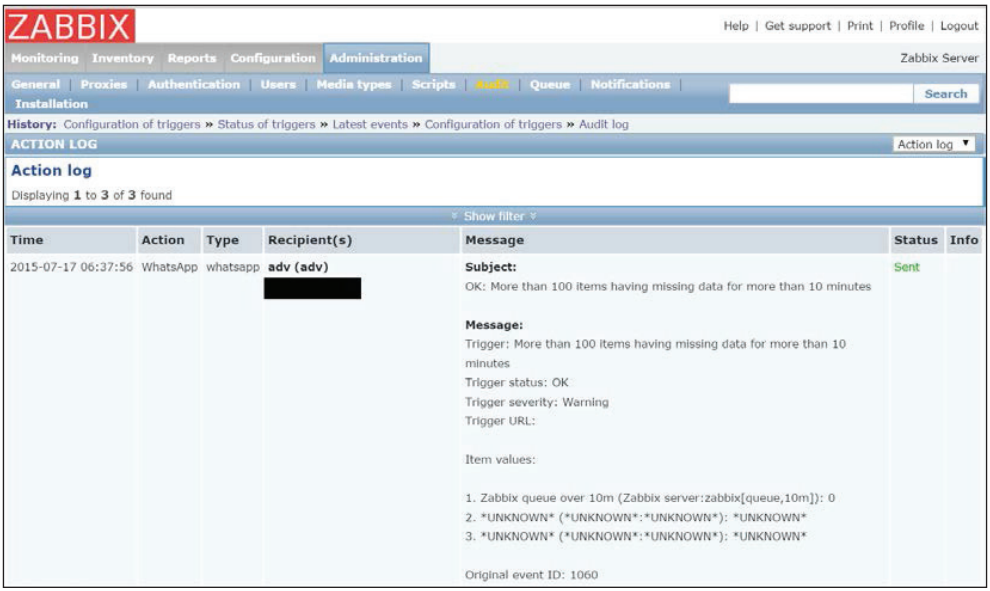


Рис. 10.10 ❖ Реакция на событие, вызванное добавлением временного правила в iptables

Здесь я добавил строку (выделена жирным), которая записывает выполняемую команду в журнал. Теперь давайте посмотрим, как отработал сценарий:

```
$ tail -n 12 /var/log/whatsapp.log

sudo -u yowsup /home/yowsup/yowsup-master/yowsup-cli demos -c /home/yowsup/yowsup-master/
yowsup.config -s 4176XXXXXXX "OK: More than 100 items having missing data for more than 10
minutes Trigger: More than 100 items having missing data for more than 10 minutes

Trigger status: OK
Trigger severity: Warning
Trigger URL:

Item values:

1. Zabbix queue over 10m (Zabbix server:zabbix[queue,10m]): 0
2. *UNKNOWN* (*UNKNOWN*:*UNKNOWN*): *UNKNOWN*
3. *UNKNOWN* (*UNKNOWN*:*UNKNOWN*): *UNKNOWN*

Original event ID: 1060"
```

Как можно видеть на рис. 10.11, команда отработала правильно, и даже многострочное сообщение было успешно доставлено – окончательная проверка всей цепочки прошла успешно.

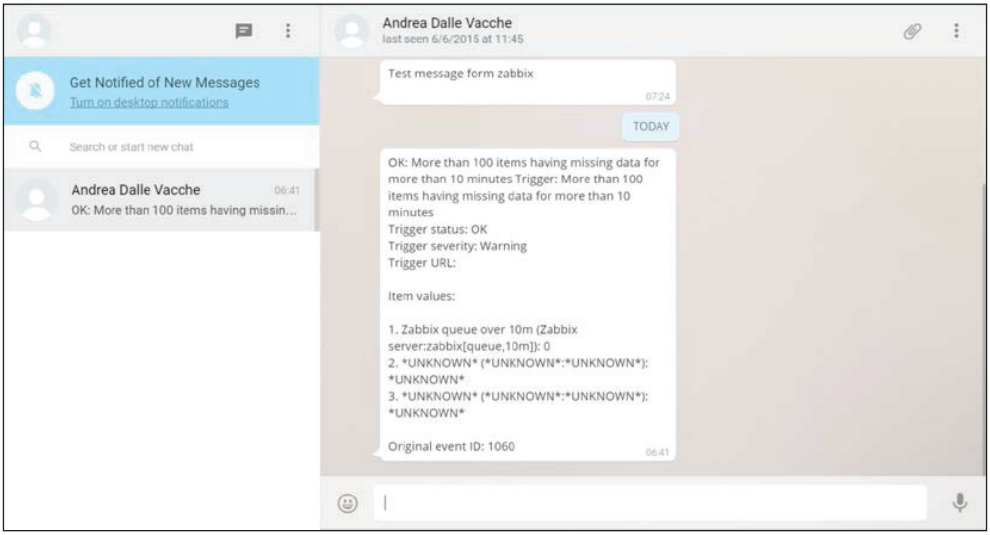


Рис. 10.11 ❖ Окончательная проверка всей цепочки прошла успешно

Такой способ интеграции по-настоящему удобен, особенно в наше время, когда многие имеют смартфоны, постоянно подключенные к сети.

Но здесь есть несколько моментов, которые необходимо учесть. Во-первых, нужно решить, посылать ли оповещение сразу всей группе или отдельным лицам. Если требуется оповестить группу целиком, следует использовать идентификатор JID группы `39340XXXXXXX-1436940409`.

То же сообщение можно послать группе `zabbix_alert`, включающей администраторов Zabbix, которая была определена выше и используется как идентификатор JID для способа оповещения WhatsApp администраторов Zabbix.

Результат отправки сообщения этой группе показан на рис. 10.12.

А теперь посмотрим, как интегрировать Zabbix с RT.

Обзор системы Request Tracker

Цитата с веб-сайта компании Best Practical:

> «*RT – это тщательно протестированная система учета и отслеживания заявок, которая используется тысячами организаций для учета проблем, заявок, обслуживания клиентов, оформления рабочих процессов, управления изменениями, выполнения сетевых операций, оказания консультационных услуг и многого другого. Организации по всему миру могут работать бесперебойно, система RT существует вот уже более 10 лет*».

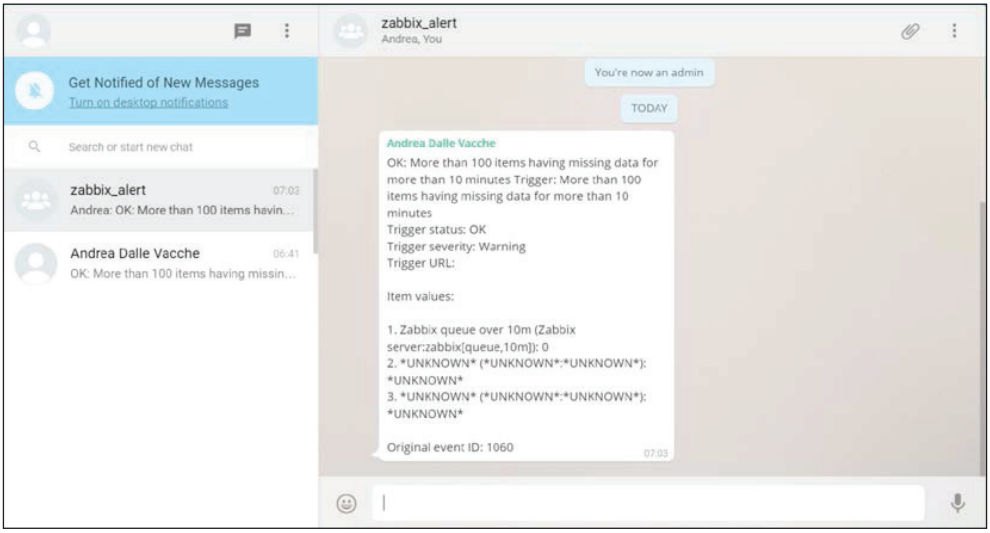


Рис. 10.12 ❖ Результат отправки сообщения группе `zabbix_alert`

Иными словами, это мощная и очень простая система с открытым исходным кодом, которую легко можно интегрировать с Zabbix. Разумеется, это не единственная система учета и отслеживания заявок, которую можно интегрировать с Zabbix; поняв основные идеи, заложенные в следующем примере, вы сможете интегрировать свою систему мониторинга с любым продуктом.

Request Tracker (RT) – это веб-приложение, написанное на Perl, которое использует веб-сервер в качестве интерфейса и реляционную базу данных для хранения информации. В основном взаимодействия с системой осуществляются через ее веб-интерфейс, но она также способна анализировать электронные письма, превращать их в зарегистрированные заявки и следить за последующим обменом электронными письмами между клиентом и персоналом, осуществляющим техническую поддержку. Наибольший интерес для нас представляет простой, но эффективный программный интерфейс REST API, который мы попробуем использовать для создания и отслеживания проблем из Zabbix. Кроме всего прочего, система RT имеет мощный механизм, способный выполнять фрагменты кода, называемые **сценариями**, который позволяет не только автоматизировать выполнение операций внутри RT, но и взаимодействовать с внешними системами с применением любого доступного протокола.

На рис. 10.13 изображена обобщенная архитектура приложения. Все данные хранятся в базе данных, а основная логика приложения способна взаимодействовать с внешним миром посредством веб-сервера, электронной почты или сценариев.

Обсуждение вопросов установки и настройки RT выходит далеко за рамки этой книги, поэтому я буду полагать, что у вас уже имеется действующий сервер RT

и несколько настроенных учетных записей и групп. Если вам требуется выполнить установку RT «с нуля», обращайтесь к инструкциям, доступным по адресу: http://www.bestpractical.com/docs/rt/4.2/README.html.

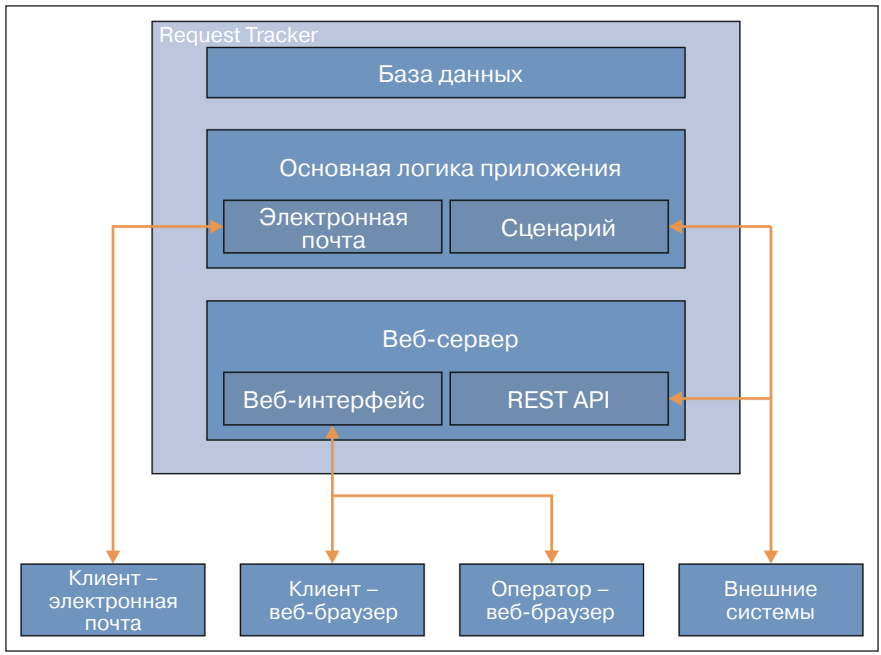


Рис. 10.13 ❖ Обобщенная архитектура Request Tracker

Настройка RT для интеграции с Zabbix

Двумя основными элементами RT являются заявки (tickets) и очереди (queues). Заявки служат для слежения за решением проблемы. Заявки имеют следующие основные пункты жизненного цикла:

- создание заявки с первичным описанием проблемы;
- оператор принимает заявку и приступает к работе над ней;
- этапы решения проблемы записываются в историю заявки;
- после окончательного решения проблемы заявка закрывается и переносится в архив.

Все метаданные заявки, такие как дата создания, продолжительность решения проблемы, пользователь, создавший заявку, оператор, работавший над ней, и др., записываются и группируются с метаданными других заявок для построения статистик и вычисления оценки качества обслуживания.

Очередь – это коллекция заявок и средство распределения заявок по разным категориям. Система позволяет определить несколько разных очередей, напри-

мер для разных отделов в организации, для разных продуктов или в соответствии с любыми другими критериями, помогающими упростить организацию заявок.

Давайте посмотрим, как настроить заявки и очереди в RT, чтобы получить возможность передавать всю необходимую информацию в Zabbix и обратно.

Создание отдельной очереди для Zabbix

Примечательной особенностью очередей является возможность настройки всех аспектов заявок, от требуемых полей до подробностей обработки, принадлежащих конкретной очереди. Соответственно, первый наш шаг – создание очереди для всех заявок, создаваемых действиями Zabbix. На этом этапе мы сможем определить основные характеристики заявок, создаваемых системой Zabbix.

Чтобы создать очередь, перейдите в раздел **Admin ⇨ Queues ⇨ Create** (Администрирование ⇨ Очереди ⇨ Создать) и заполните поля формы. Для наших целей достаточно определить имя очереди и необязательное описание, как показано на рис. 10.14.

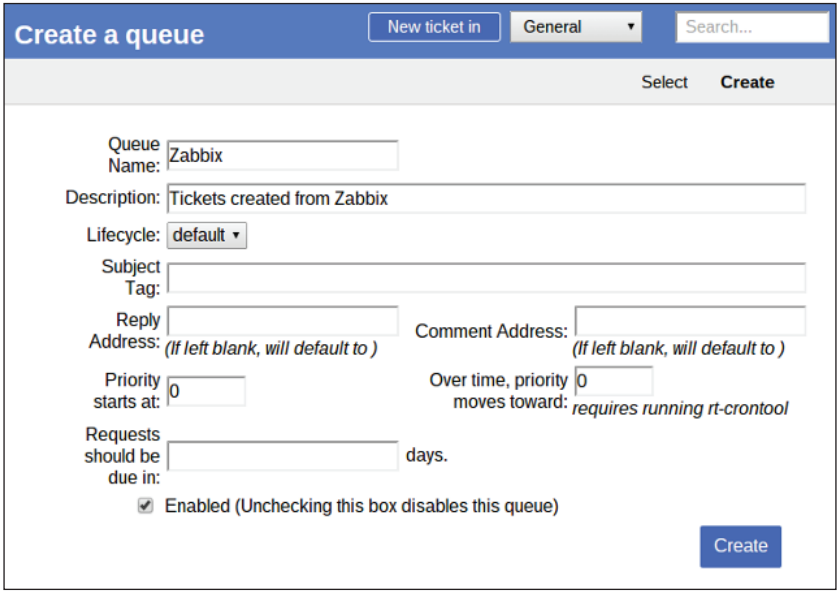


Рис. 10.14 ❖ Достаточно определить имя очереди и необязательное описание

После создания очереди можно приступать к дальнейшим настройкам. Для этого перейдите в раздел **Admin ⇨ Queues ⇨ Select** (Администрирование ⇨ Очереди ⇨ Выбор) и выберите очередь Zabbix. Здесь вы должны наделить правами доступа группы пользователей (или хотя бы отдельных пользователей), чтобы ваш персонал мог работать с заявками, созданными системой Zabbix. Возможно, у вас появится желание добавить собственные поля, о чем, собственно, мы поговорим ниже.

А теперь давайте посмотрим, какие разделы в заявке могут представлять для нас интерес с точки зрения интеграции.

Настройка заявок – раздел «Ссылки»

Учитывая, что наша главная цель – интеграция действий и событий Zabbix с RT, наибольший интерес для нас представляет раздел **Links** (Ссылки) заявки. Как следует из названия, в этом разделе можно определить ссылки на другие заявки или системы. В этом разделе можно вставлять полезные ссылки в ходе создания заявки или ее редактирования, как показано на рис. 10.15.

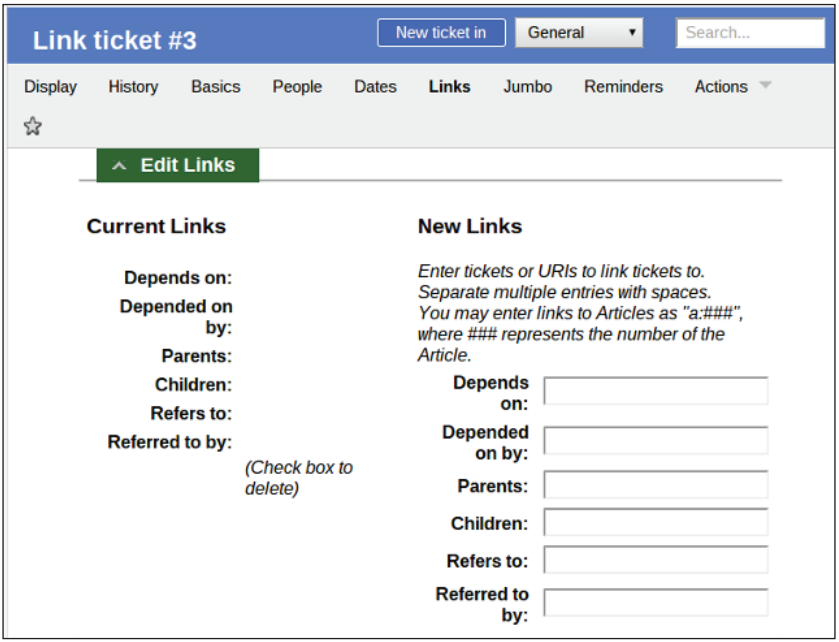


Рис. 10.15 ❖ Раздел **Links** (Ссылки) заявки

Как вы уже наверняка поняли, мы будем использовать поле **Refers to:** (Ссылается на:) для определения ссылки на событие Zabbix, создавшее заявку. Как будет показано далее, поле квитирования события, в свою очередь, будет ссылаться на соответствующую заявку, чтобы легко можно было переходить из одной системы в другую и следить за происходящим.

Настройка заявок – приоритет заявки

Еще одно интересное поле – приоритет заявки – находится в разделе **Basics** (Основные). Это целочисленное значение в диапазоне от 0 до 100, позволяющее сортировать заявки по уровню их важности. В RT отсутствуют какие-либо требования к определению приоритетов, поэтому мы можем использовать уровни важности,

используемые триггерами Zabbix. Это означает, что если вы решите хранить в заявке информацию о важности триггера, у вас на выбор есть два варианта:

- игнорировать поле приоритета в заявке и определить свое поле, отражающее важность триггера в виде текста;
- определить соответствие между уровнями важности триггеров и приоритетами заявок и использовать его для сохранения уровня важности при создании заявки.

Единственное преимущество первого варианта – удобочитаемость заявок, так как простого взгляда будет достаточно, чтобы определить важность соответствующего триггера. С другой стороны, второй вариант позволяет сортировать заявки по приоритету и решать сначала наиболее важные или неотложные проблемы.

В этом примере мы пойдем вторым путем и при создании заявки будем определять приоритеты, пользуясь соотношением в табл. 10.1.

Таблица 10.1 ❖ Соответствие приоритетов заявок и уровней важности триггеров

Метка со значением важности триггера	Значение важности триггера	Приоритет заявки
Not classified	0	0
Information	1	20
Warning	2	40
Average	3	60
High	4	80
Disaster	6	100

Больше никаких настроек на стороне RT не требуется. Это соотношение охватывает весь диапазон значений приоритетов, поэтому заявки Zabbix будут правильно сортироваться не только в специализированной очереди, но и в системе RT в целом.

Настройка заявок – собственные поля

Как мы уже видели в *главе 6* «*Управление оповещениями*», действия Zabbix имеют доступ к большому количеству макросов и, как следствие, к большому объему информации о своем событии. Было бы совершенно логично привести эту информацию к удобочитаемому виду и добавить ее в отправляемое электронное письмо с применением собственных полей заявки в RT, а не ограничиваться одним только ее описанием.

Одним из больших достоинств собственных полей является их доступность функциям поиска и фильтрации, как если бы они были встроенными полями. Это означает, что если поместить в дополнительное поле имя хоста, связанного с событием, вы сможете отыскать все заявки, принадлежащие определенному хосту. Поэтому давайте добавим в заявки, включаемые в очередь Zabbix, пару своих полей для хранения информации, которые потом пригодятся для поиска. Перейдите

в раздел **Admin ⇨ Custom Fields ⇨ Create** (Администрирование ⇨ Собственные поля ⇨ Создать) и создайте поле «Hosts», как показано на рис. 10.16.

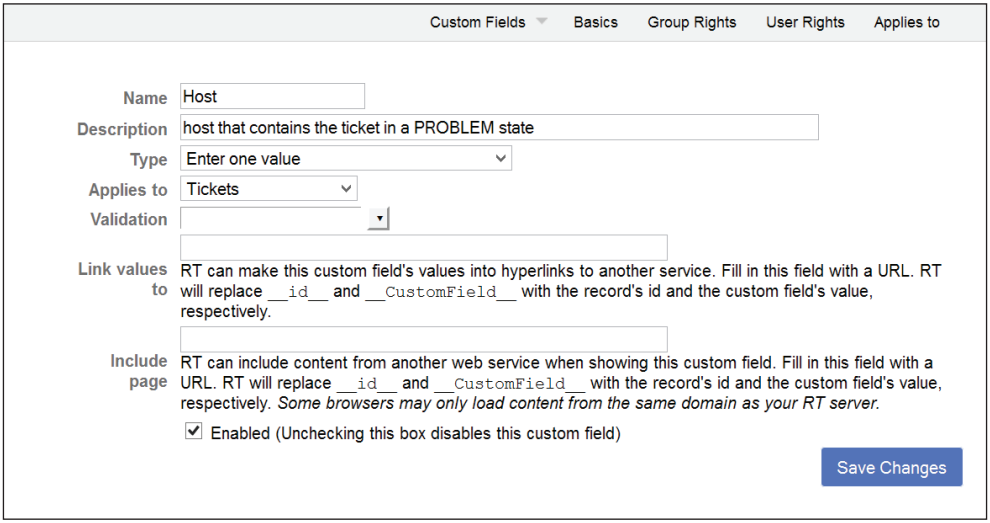


Рис. 10.16 ❖ Определение собственного поля «Hosts»

Выберите в поле **Type** (Тип) значение **Enter multiple values** (Ввод нескольких значений). Это позволит указывать несколько хостов для сложных триггеров, использующих элементы данных с разных хостов.

Аналогично можно определить поля для названий триггеров, элементов данных и ключей. Покончив с этим, необходимо связать эти поля с заявками в очереди Zabbix. Выберите очередь Zabbix в разделе **Admin ⇨ Queues ⇨ Select** (Администрирование ⇨ Очереди ⇨ Выбор) и в форме **Tickets** (Заявки) перейдите в раздел **Custom fields ⇨ Tickets** (Собственные поля ⇨ Заявки). Выберите поля, которые предполагается связать с заявками, как показано на рис. 10.17.

Рис. 10.17 ❖ Выбор полей для связывания с заявками

По окончании поля должны появиться во всех заявках в очереди Zabbix (рис. 10.18).

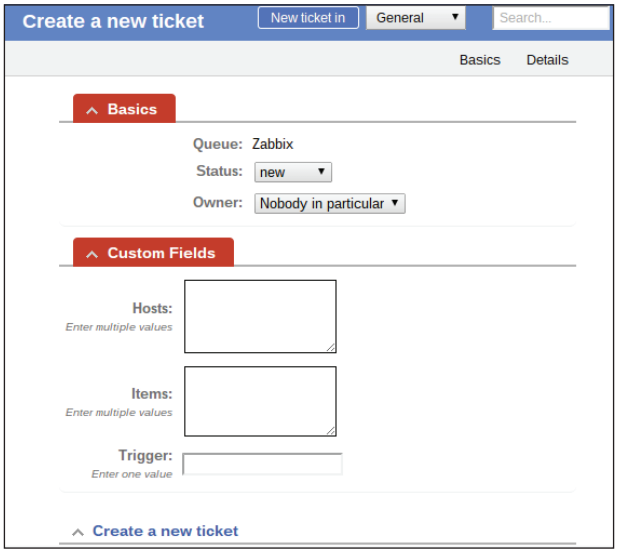


Рис. 10.18 ❖ Поля появились во всех заявках в очереди Zabbix

При необходимости можно создать любое количество дополнительных полей для хранения имен хостов, событий, IP-адресов, собственных макросов и т. д. Вы сможете выполнять поиск по любому из них в очереди Zabbix, в веб-интерфейсе RT. Дополнительные поля появятся, как и следовало ожидать, внизу формы поиска (рис. 10.19).

Соединение с Request Tracker API

Система RT открывает доступ к REST-подобному API по протоколу HTTP. То есть прикладной интерфейс системы легко можно исследовать с помощью таких инструментов, как `wget` и `netcat`. Давайте воспользуемся этой возможностью, чтобы разобраться с некоторыми особенностями интерфейса, прежде чем приступить к использованию библиотеки на языке Python.

Базовый URL прикладного интерфейса RT API имеет вид: `../REST/1.0/`. То есть если сам сервер имеет URL: `http://your.domain.com/rt`, тогда его прикладной интерфейс будет доступен по адресу: `http://your.domain.com/rt/REST/1.0/`. При попытке соединиться с ним вы должны получить сообщение, требующее указать учетные данные (для простоты из следующего листинга были убраны некоторые заголовки):

```
$ ncat example.com 80
GET /rt/REST/1.0/ HTTP/1.1
```

```
Host: example.com
HTTP/1.1 200 OK
[...]
Content-Type: text/plain; charset=utf-8

22
RT/4.2.0 401 Credentials required
```

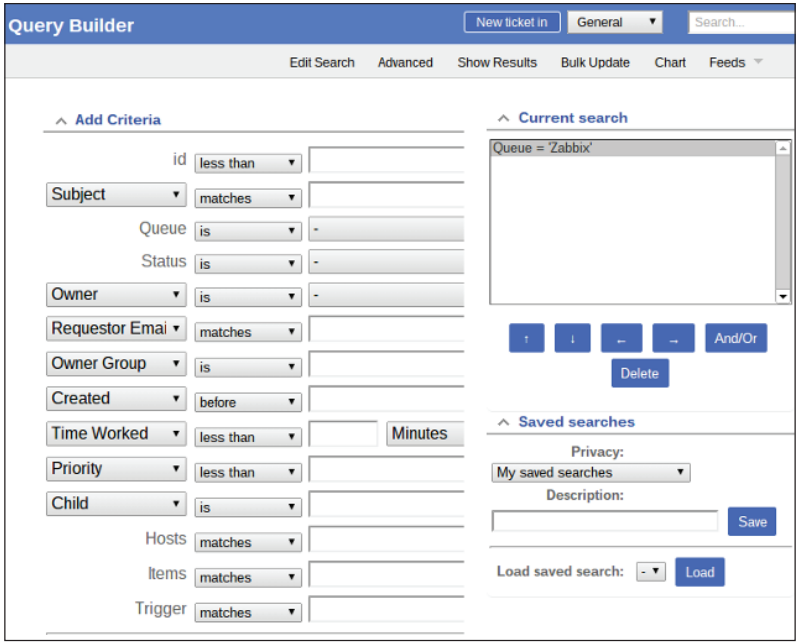


Рис. 10.19 ❖ Дополнительные поля внизу формы поиска

Прикладной интерфейс не имеет своего механизма аутентификации, отдельного от самого приложения, поэтому лучший способ аутентифицироваться – получить сеансовый cookie в форме входа и использовать его во всех запросах к API. Получить cookie можно с помощью `wget`:

```
$ wget --keep-session-cookies --save-cookies cookies.txt --post-data
'user=root&pass=password' http://example.com/rt/
```

Эта команда сохранит сеансовый cookie в файле `cookies.txt`, например:

```
$ cat cookies.txt
# HTTP cookie file.
# Generated by Wget on 2015-07-10 10:16:58.
# Edit at your own risk.

localhost FALSE /rt FALSE 0 RT_SID_example.com.80
2bb04e679236e58b406b1e554a47af43
```

Теперь, имея действительный сеансовый cookie, можно выполнять запросы к API. Ниже приводится пример GET-запроса к общей очереди:

```
$ ncat localhost 80
GET /rt/REST/1.0/queue/1 HTTP/1.1
Host: localhost
Cookie: RT_SID_example.com.80=2bb04e679236e58b406b1e554a47af43

HTTP/1.1 200 OK
[...]
Content-Type: text/plain; charset=utf-8

RT/4.2.0 200 Ok

id: queue/1
Name: General
Description: The default queue
CorrespondAddress:
CommentAddress:
InitialPriority: 0
FinalPriority: 0
DefaultDueIn: 0
```

Как видите, с API легко взаимодействовать без специального кодирования или декодирования. Однако в нашем случае еще проще использовать библиотеку, избавляющую от необходимости анализа каждого HTTP-запроса. Rtkit – библиотека для Python 2.x – облегчает работу с API из программ на языке Python. Она позволяет посылать запросы и получать ответы с применением обычных для Python структур данных. Установить библиотеку можно с помощью диспетчера pip:

```
$ pip install python-rtkit
```

Здесь предполагается, что сам диспетчер pip уже установлен. Если это не так, установите его командой:

```
$ yum install -y python-pip
```

После установки библиотека станет доступной для импортирования в виде нескольких модулей. Давайте посмотрим, как выполнить те же взаимодействия, что и выше (аутентификация и запрос общей очереди), из сеанса интерактивной оболочки Python 2.x:

```
$ python2
Python 2.7.5 (default, Sep 6 2013, 09:55:21)
[GCC 4.8.1 20130725 (prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from rtkit.resource import RTResource
>>> from rtkit.authenticators import CookieAuthenticator
>>> from rtkit.errors import RTResourceError
>>>
>>> res = RTResource('http://localhost/rt/REST/1.0/', 'root', 'password',
```

```
CookieAuthenticator)
>>>
>>> response = res.get(path='queue/1')
>>> type(response)
<class 'rtkit.resource.RTResponse'>
>>> type(response.parsed)
<type 'list'>
>>> response.parsed
[[('id', 'queue/1'), ('Name', 'General'), ('Description', 'The default queue'),
('CorrespondAddress', ''), ('CommentAddress', ''), ('InitialPriority', '0'),
('FinalPriority', '0'), ('DefaultDueIn', '0')]]
```

Как видите, ответ преобразуется в список кортежей со всеми атрибутами объекта RT.

Теперь, когда у нас есть своя очередь и определены дополнительные поля для заявок Zabbix, мы можем организовать взаимодействие с RT API из сценариев на языке Python.

На этом процесс настройки на стороне RT закончен. Мы уже фактически интегрировали действия Zabbix и заявки RT.

Настройка Zabbix для интеграции с Request Tracker

Наша цель – определить шаг в действии Zabbix, который:

- создаст заявку с информацией о событии;
- добавит в заявку ссылку на событие Zabbix, сгенерировавшее ее;
- добавит в событие ссылку на только что созданную заявку.

Если первый пункт можно реализовать простой операцией отправки электронного письма в адрес RT, то для реализации двух других придется написать свой код. Для этого лучше всего определить в Zabbix новый способ оповещения в форме сценария, который будет выполнять следующие действия:

- примет текст сообщения;
- извлечет из него необходимую информацию;
- создаст заявку со всеми нестандартными полями, заполнит их и установит ссылки;
- получит идентификатор заявки;
- запишет ссылку на созданную заявку в поле квитирования события.

Но, прежде чем приступать к разработке сценария, определим новый способ оповещения и привяжем его к пользователю (вы можете установить эту связь с любым пользователем, но в данном примере мы будем использовать отдельную учетную запись `rt_tickets`, как показано на рис. 10.20).

Связывая способ оповещения с пользователем, в поле **Send to** (Отправлять на) следует указать базовый адрес URL системы RT, чтобы не определять его статически в тексте сценария. Это отражено на рис. 10.21.

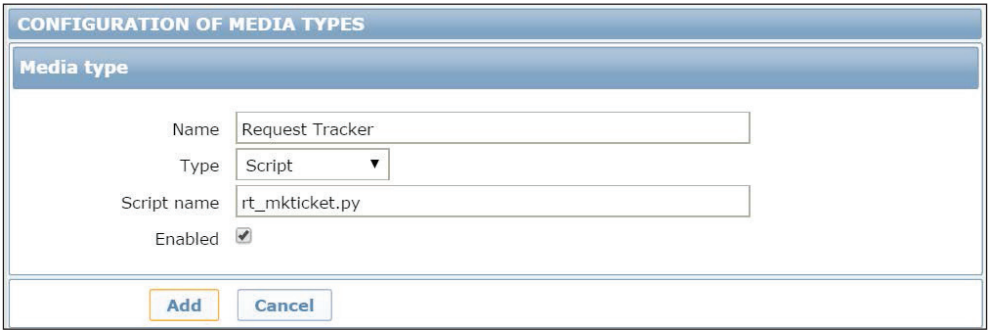


Рис. 10.20 ❖ Определение нового способа оповещения

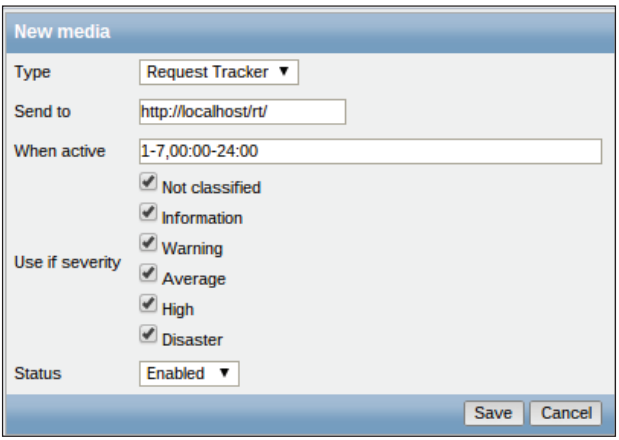


Рис. 10.21 ❖ Настройка базового адреса URL системы RT

После сохранения вся информация будет доступна для обзора на вкладке **Media** (Оповещения), как показано на рис. 10.22. За полем с адресом следует поле, определяющее периоды времени для данного способа оповещения, а затем – шестибуквенный код, отражающий уровни важности. Если отключить один из них, соответствующая буква будет окрашена в серый цвет.

Теперь создадим шаг в действии, реализующий отправку сообщения пользователю `rt_tickets` с использованием нового способа оповещения. Важно заметить, что пользователь `rt_tickets` не будет фактически получать сообщений, потому что сценарий в действительности создает заявку в системе RT, но все это выполняется совершенно прозрачно с точки зрения действия Zabbix. Вы можете вложить в сообщение любую информацию, но, как минимум, должны указать в теме имя триггера и идентификатор события, а в теле – важность, имена хостов и элементов, чтобы сценарий смог извлечь ее и заполнить соответствующие поля заявки. Это отражено на рис. 10.23.

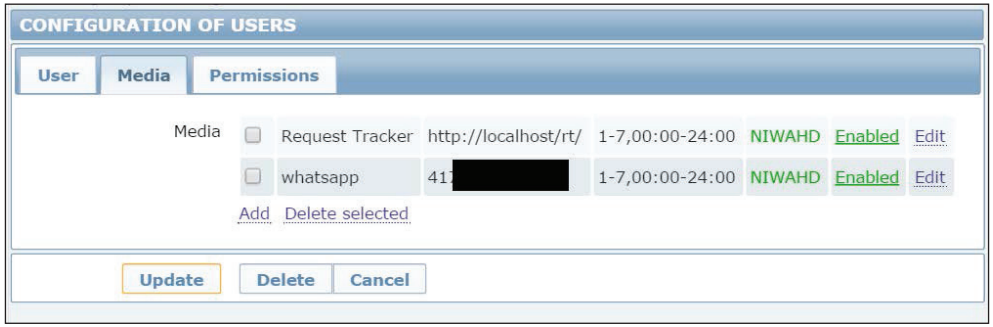


Рис. 10.22 ❖ После сохранения информация доступна для обзора на вкладке **Media** (Оповещения)

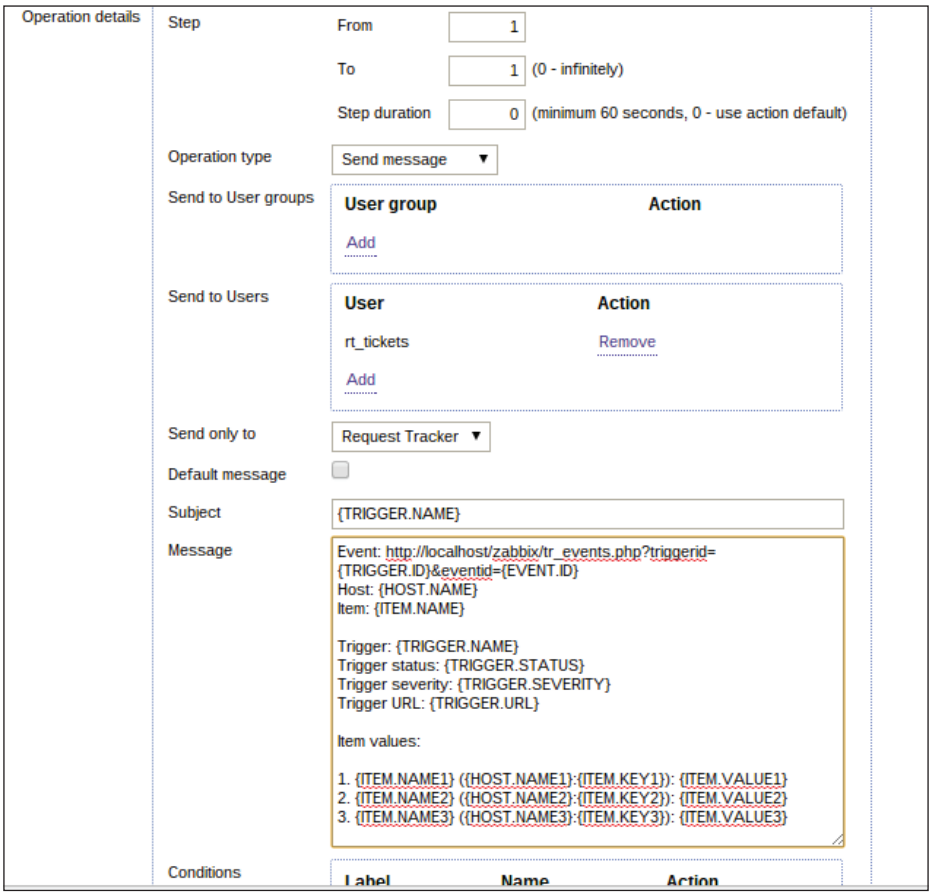


Рис. 10.23 ❖ Определение сообщения

Теперь можно приступать к разработке сценария и его использованию для создания заявок из Zabbix.

Создание заявок RT из событий Zabbix

Поиск сценариев, реализующих операции, система Zabbix ищет в каталоге, объявленном в параметре `AlertScriptsPath` в файле `zabbix_server.conf`. По умолчанию используется каталог `${datadir}/zabbix/alertscripts`, или в Red Hat: `/usr/lib/zabbix/alertscripts/`.

Именно в этот каталог мы поместим сценарий `rt_mkticket.py`. Действие Zabbix, настроенное выше, будет вызывать этот сценарий с тремя параметрами в следующем порядке:

- получатель;
- тема;
- сообщение.

Как вы уже знаете, тема и само сообщение определяются в операции действия и идентифицируют событие, вызвавшее действие. Получатель определяется в настройках способа оповещения пользователя, принимающего сообщение, и обычно имеет вид адреса электронной почты. Но в данном случае это будет базовый адрес URL системы Request Tracker.

Итак, начнем сценарий с импортирования необходимых библиотек и парсинга аргументов:

```
#!/usr/bin/python2
from pyzabbix import ZabbixAPI
from rtkit.resource import RTResource
from rtkit.authenticators import CookieAuthenticator
from rtkit.errors import RTResourceError
import sys
import re

lines = re.findall(
    r'^(?!(Host:|Event:|Item:|Trigger severity:))(.*)$',
    message, re.MULTILINE)
desc = '\n'.join([y for (x, y) in lines])

rt_url = sys.argv[1]
rt_api = rt_url + 'REST/1.0/'
trigger_name = sys.argv[2]
message = sys.argv[3]
```

Затем извлечем из сообщения ссылку на событие, важность триггера, список имен хостов и список имен элементов данных. Для этого можно использовать мощные функции Python для работы с регулярными выражениями:

```
event_id = re.findall(
    r'^Event: (.+)$', message, re.MULTILINE)[0]
severity = re.findall(
```

```
    r'^Trigger severity: (.+)$', message, re.MULTILINE)[0]
hosts = re.findall(r'^Host: (.+)$', message, re.MULTILINE)

items = re.findall(r'^Item: (.+)$', message, re.MULTILINE)
lines = re.findall(
    r'^(?!(Host:|Event:|Item:|Trigger severity:))(.*)$',
    message, re.MULTILINE)

desc = '\n'.join([y for (x, y) in lines])
```

Идентификатор события должен быть уникальным, триггер может ссылаться на несколько элементов данных и на несколько хостов. Для построения списка хостов предыдущий фрагмент отыскивает все строки, начинающиеся с `Host:`. В объявлении сообщения выше указана только одна строка `Host: {HOST.NAME}`. Это было сделано ради улучшения читаемости примера, но в действительности шаблон может содержать несколько таких строк (только не забудьте использовать макросы `{HOST.NAME1}`, `{HOST.NAME2}`, `{HOST.NAME3}` и т. д., иначе во всех строках будет повторяться одно и то же имя хоста). То же относится к именам элементов данных. Вслед за этим извлекаются остальные строки сообщения и объединяются в одну строку.

Для определения важности триггера используется макрос `{TRIGGER.SEVERITY}`, который будет замещаться строкой с описанием, а не числовым значением. Поэтому необходимо определить простой словарь, отображающий метки с описанием уровня важности в значения приоритетов RT, как описывалось выше в этой главе:

```
priorities = {
    'Not classified': 0,
    'Information': 20,
    'Warning': 40,
    'Average': 60,
    'High': 80,
    'Disaster': 100 }
```

Нам также необходимо заранее знать имя очереди для создания заявок или, еще лучше, ее идентификационный номер:

```
queue_id = 3
```

Теперь у нас есть все необходимое для оформления запроса на создание новой заявки и его отправки в систему Request Tracker:

```
ticket_content = {
    'content': {
        'Queue': queue_id,
        'Subject': trigger_name,
        'Text': desc,
        'Priority': priorities[severity],
        'CF.{Hosts}': ','.join(hosts),
        'CF.{Items}': ','.join(items),
        'CF.{Trigger}': trigger_name
```

```
    }
}

links = {
    'content': {
        'RefersTo': event_url
    }
}
```

Сначала создаются два словаря: один – для основного содержимого заявки, а второй – для раздела ссылок, который должен заполняться отдельно.

Далее следует основная часть сценария: в первую очередь необходимо аутентифицироваться в RT API (подставьте в текст сценария свои фактические учетные данные!), потом создать новую заявку, получить ее идентификатор и ввести ссылку на страницу события в Zabbix:

```
rt = RTResource(rt_api, 'root', 'password', CookieAuthenticator)
ticket = rt.post(path='ticket/new', payload=ticket_content,)
(label,ticket_id) = ticket.parsed[0][0]
refers = rt.post(path=ticket_id + '/links', payload=links,)
```

Вот почти и все, осталось лишь квитировать событие в Zabbix ссылкой на только что созданную заявку:

```
event_id = re.findall(r'eventid=(\d+)', event_url)[0]
ticket_url = (rt_url + 'Ticket/Display.html?id=' +
              ticket_id.split('/')[1])
print(ticket_url)
zh = ZabbixAPI('http://localhost/zabbix')
zh.login(user='Admin', password='zabbix')
ack_message = 'Ticket created.\n' + ticket_url
zh.event.acknowledge(eventids=event_id, message=ack_message)
```

Последний фрагмент очень прост. После извлечения `event_id` и создания адреса URL заявки он подключается к Zabbix API и записывает ссылку в поле `acknowledge` события, замыкая круг.

Закончив разработку сценария, не забудьте сделать его владельцем пользователя `zabbix` и установить разрешение на выполнение:

```
$ chown zabbix rt_mkticket.py
$ chmod +x rt_mkticket.py
```

В следующий раз, когда условие действия в вашей системе вернет `true`, операция вызовет сценарий с параметрами, перечисленными ранее. Сценарий создаст заявку со ссылкой на событие и квитирует событие ссылкой на заявку.

На рис. 10.24 приводится пример такого события. Ссылка в поле `acknowledgement` события указывает на заявку.

На рис. 10.25 показана соответствующая заявка. Поле **Refers to:** (Ссылается на:) содержит ссылку на событие, изображенное на рис. 10.24, а раздел **Custom Fields** (Собственные поля) хранит список хостов, элементов и сведения о триггере.

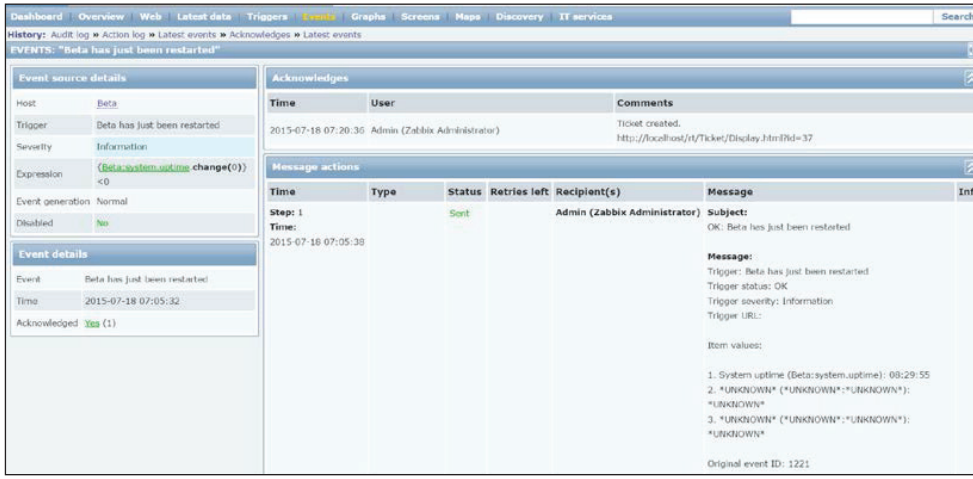


Рис. 10.24 ❖ Пример события

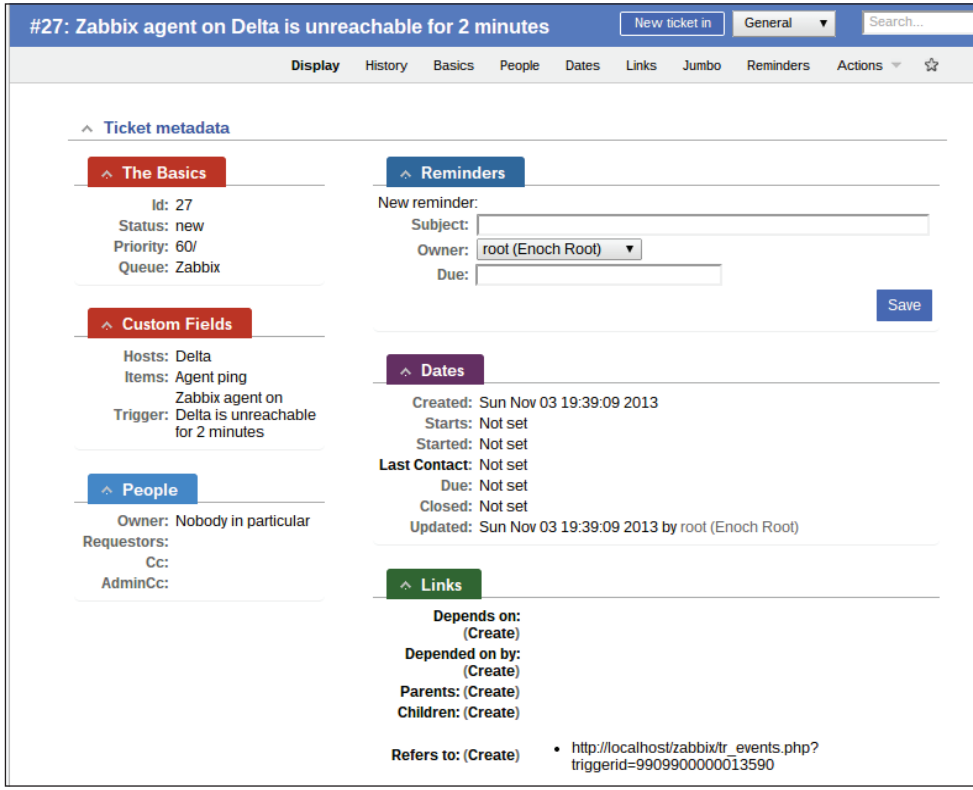


Рис. 10.25 ❖ Заявка, соответствующая событию на рис. 10.24

Сценарий реализован по аналогии со сценариями в *главе 9 «Расширение Zabbix»*. Он предназначен исключительно для демонстрации основных идей, поэтому основное внимание при его создании уделялось простоте и удобочитаемости, а не надежности и функциональности. Если вы соберетесь использовать его в действующем окружении, добавьте все необходимые проверки на ошибки.

В заключение

Мы достигли конечной точки путешествия по системе мониторинга Zabbix. В этой книге вы узнали, как спроектировать и реализовать общую архитектуру мониторинга; как создавать гибкие и эффективные элементы данных, триггеры и действия; как лучше визуализировать данные. Вы также узнали, как реализуются нестандартные агенты, исследовав протоколы Zabbix, и как написать код, управляющий всеми аспектами Zabbix посредством API. В этой главе мы лишь приоткрыли возможности интеграции Zabbix с окружающей инфраструктурой. В действительности эти возможности намного шире, чем было показано, в том числе получение и изменение информации о пользователях и группах с применением системы управления идентификацией, получение инвентарной информации из системы управления ресурсами, передача инвентарной информации в базу данных CMDB (Configuration management database – база данных управления конфигурацией), и многое другое. После изучения шагов, необходимых для интеграции Zabbix с системой управления заявками и внешними средствами оповещения, вы узнали, как подготовить две системы, чтобы они могли взаимодействовать и обмениваться данными посредством прикладных программных интерфейсов друг друга. Попутно мы рассмотрели и проанализировали все важнейшие аспекты безопасности, а также узнали, как ослабить вероятные риски, которые влечет за собой внедрение системы мониторинга. В настоящий момент вы имеете все необходимые знания, чтобы реализовать и развернуть отдельную и безопасную систему мониторинга.

Я надеюсь, что со вновь полученными знаниями и навыками вы сможете использовать весь потенциал системы мониторинга Zabbix и превратить ее в важнейший ресурс своей инфраструктуры. Ваши силы и время, затраченные на этом пути, окупятся сторицей.

Предметный указатель

Z

A

Б

В

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным
платежом, выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Андреа Далле Вакке

Zabbix. Практическое руководство

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Киселев А. Н.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 33. Тираж 200 экз.

Веб-сайт издательства: www.дмк.рф