**BC/NW 2018 № 2 (33):4.1**

**АВТОМАТИЗАЦИИ РЕШЕНИЯ ЗАДАЧ АДМИНИСТРАТОРА КОРПОРАТИВНОЙ СЕТИ**

Данилин Г.Г., Фомичёв Н.А.

 Предварительные замечания

 Современные компьютерные сети могут состоять из десятков тысяч серверов. Каждый из которых может содержать несколько десятков файлов конфигураций, используемых работающими на нем приложениями. Системным администраторам при этом необходимо отслеживать, исправлять и документировать все изменения и зависимости в работе таких приложений.

 Такого рода задачи решаются с помощью систем управления конфигурациями, англоязычный термин *Configuration Management*. Впервые такой подход был использован в 1960 военно-воздушными силами США и опубликованы стандарты *MIL-STD-480* и *MIL-STD-481*. Которые в последствии легли в основу таких широко используемых сейчас стандартов как *systems engineering* (SE), *COBIT*, *Information Technology Infrastructure Library* (ITIL), *ISO* 9000.

 Наибольшее распространение системы управления конфигурациями получили в крупных информационных системах, которые поддерживают работу современных социальных сетей и популярных интернет порталов.

По опубликованным Facebook данным в 2012 году их сеть насчитывала 180 000 серверов, при этом в 2010 году их было 60 000, а в 2009 всего 30 000.

Системы управления конфигурациями также широко востребованы военной индустрией, но из открытых источников доступны только опубликованные стандарты.

 Объектом исследования в статье является корпоративная сеть передачи данных. Предметом исследования является автоматизация **части** задачи системного администратора такой сети.

 Целью работы является определение подмножества задач, решаемых системным администратором, их анализ и автоматизация средствами системы управления конфигурациями.

 Система управления конфигурациями puppet

Система управления конфигурациями puppet была разработана в 2005 году на языке программирования *Ruby*. Принадлежит к классу *open source* программ, то есть ее исходный код доступен для скачивания и модификации. Puppets выпускается в 2 версиях, *opensource* и *enterprise*. *Opensource* версия является бесплатной для использования и не имеет платной поддержки от разработчиков, в случае наличия проблем с ее работой. *Enterprise* версия *puppet* включает в себя весь функционал *opensource* версии, графический интерфейс пользователя, а так же функционал для управления виртуальными серверами *Vmware*, *Amazon*, но при этом не является бесплатной работы .

*Puppet* использует клиент-серверную модель взаимодействия, схема представлена на Рис. 1.



 Рис. 1 Схема взаимодействия puppet

 Клиенты - это специальное программное обеспечение, которое устанавливается на каждый сервер сети. Минимальные системные требования к серверу для установки такого клиента это интерпретатор языка *Ruby* версии не ниже 1.9.3, программные библиотеки *Facter* версии не ниже 1.6.2 и *Hiera* версии 1.0. На сервере должно быть не менее 512 МБ памяти. Каждый клиент *puppet* после установки и настройки подключается по протоколу TCP/IP к основному серверу *puppet*. На прикладном уровне модели OSI используется протокол взаимодействия *HTTP*. Возможно написание собственных клиентов *puppet*, так как доступен программный *API*. Он основан на работе с *REST* коллекциями, уникальными идентификаторами которых является *WEB URL*.

Работающий на сервере клиент *puppet* представлен процессом *puppetd*.

Сервер *puppet*, обрабатывающий соединения клиентов имеет такие же системные требования как и клиент. Может запускаться как отдельный процесс в ОС или работать под управлением *WEB* серверов *apache*, *nginx*, *mongrel*. На сервере *puppets* находятся описания на специальном декларативном языке программирования. Эти правила загружаются клиентами *puppet* с центрального сервера и обрабатываются каждым клиентом локально.

Декларативный язык программирования в puppet снабжен набором методов для решения часто возникающих задач системного администратора. Например, для проверки и установки записи для *localhost.localdomain* в файл *hosts*, описание будет следующим:

host { 'localhost.localdomain':

 ensure => 'present',

 target => '/etc/hosts',

 ip => '127.0.0.1',

 host\_aliases => ['localhost']}

Данный код при запуске на клиенте *puppet* проверит, что если существует файл */etc/hosts*, то в нем должна быть запись вида

*127.0.0.1 localhost localhost.localdomain*

Если такой записи нет, то она будет добавлена, если запись в файле присутствует, то файл не изменится.

Проверка наличия в системе файла и его прав доступа для *puppet* будет выглядеть следующим образом

*file { "/etc/passwd":*

 *ensure => "present",*

 *owner => "root",*

 *group => "root",*

 *mode => 644,*

*}*

Puppets проверит, что файл существует и его владельцем является пользователь *root*, с группой *root*, а права доступа 644. Если это не так, то puppets выставит соответствующих пользователя, группу и права доступа.

 Для управления файлами конфигурации в puppet используется подход, заключающийся в копировании либо файла конфигурации, либо его шаблона с сервера *puppet*. Это оправдано, если *puppet* используется и управляет всеми серверами. Обычно же стоит задача внедрения системы управления конфигурациями. В таком случае если *puppet* установлен на достаточно большом количестве серверов, то задача сбора с них определенного файла конфигурации и проверка его единообразия на всех системах является крайне трудоемкой. Эта же задача может быть решена проверкой наличия нужной строки в файле и ее добавления в случае отсутствия или исправления. В *puppet* такой функционал отсутствует. В качестве обходных вариантов могут использоваться скрипты с использованием утилит *Unix*, таких как *sed, awk* или собственные программные библиотеки с методами редактирования. Например, *puppet-postfix* позволяет указать какие именно параметры необходимо задать в файле конфигурации *main.cf* для почтового сервера

*Postfix*. Такой функционал реализован через использование утилиты *postconf*, на вход которой передаются заданные в *puppet* параметры. Требуется разработка подобного рода методов для редактирования каждого файла конфигурации в системе.

 Система управления конфигурациями puppets используется в таких крупных компаниях как *Dell, Disney, Oracle, Google*. Преимуществом *puppet* в сравнении с другими системами управления конфигурациями является простота разработки описаний для запуска на клиентах и легко модифицируемый исходный код на *Ruby*. *Puppet* одна из немногих систем управления конфигурациями, которая может взаимодействовать напрямую с серверами приложений *Ruby on Rails* и их окружения ввиде набора *gem* пакетов.

 Система управления конфигурациями ansible

Система управления конфигурациями *ansible* разработана в 2011 году на языке *python*. Так же как и *puppet* относится к классу *open source* систем с доступным для скачивания исходным кодом. *Ansible* использует клиент-серверную модель взаимодействия. Клиентом является любой сервер сети, на который можно попасть по протоколу *SSH*. Такой подход позволяет управлять любым набором операционных систем, которые поддерживают доступ по протоколу *SSH.* В отличие от *puppet*, *ansible* не требует установки дополнительного ПО на сервера сети.

 В *ansible* по протоколу *SSH* передаются команды и данные в формате *JavaScript Object Notation(JSON).* Это дает возможность писать практически на любом языке программирования собственные модули, которые будут запускаться на серверах сети. На Рис. 2. представлена структура взаимодействия сервера ansible и управляемых серверов сети.



 Рис. 2. Взаимодействие сервера ansible с управляемыми серверами

У ansible для описания желаемого состояния конфигурации сервера используется язык разметки *yet another markup language* (YAML), который сами разработчики называются *playbooks*. Для работы ansible на сервер необходимо установить интерпретатор python версии не ниже 2.6, а также его модули для работы с YAML.

 Решение задачи с добавлением в */etc/hosts* строки

*127.0.0.1 localhost localhost.localdomain*

средствами ansible выглядит следующим образом

*lineinfile: dest=/etc/hosts regexp=^ \*127.0.0.1 line=127.0.0.1 localhost localhost.localdomain*

Но для того чтобы *ansible* смог бы редактировать файл */etc/hosts*, целевой сервер должен быть доступен по протоколу *SSH*, проблемы с сетью передачи данных могут стать проблемой в работе *ansible*. Для установки владельца и прав доступа к файлу в *ansible* необходимо использовать следующий код

*file: path=/etc/passwd owner=root group=root mode=0644*

 Систему управления конфигурациями *ansible* используют такие компании, как *Hewlett-Packard, Fedora Project*. Из достоинств можно выделить простоту написания правил для редактирования конфигураций и отсутствие необходимости устанавливать специальное ПО на целевые сервера. Главные недостатком является зависимость исполнения операций на серверах сети от бесперебойной работы сервера *ansible*. Потеря доступа к сети на сервере *ansible* приведет к невозможности производить какие-либо изменения на целевых серверах сети.

 Система управления конфигурациями cfengine

 Система управления конфигурациями *cfengine* является старейшей из таковых, она написана в 1993 Марком Бургесом, сотрудником университета Осло в Норвегии. Для ее разработки использован язык *C*, это одна из причин почему она наименее ресурсотребовательна по сравнению с конкурентами. Доступна в платной и бесплатной версии, на момент написания работы последняя версия была 3.4.0. Платная версия отличается от бесплатной наличием поддержки и обширного функционала управления виртуальными серверами *vmware, xen, kvm* и создания отчетов.

 *Cfengine* использует клиент-серверную модель взаимодействия, при этом наличие центрального сервера не обязательно. Схема взаимодействия представлена на Рис. 3.



 Рис. 3 Схема взаимодействия серверов под управлением *cfengine*

Возможно взаимодействие одного клиента с несколькими другими для обновления локальной копии правил. Также возможно использование клиента без каких-либо сетевых взаимодействий, для этого на системе должен присутствовать файл *promises.cf* с описанием на декларативном языке программирования того состояния системы к которому ее нужно привести.

 Клиента cfengine необходимо установить на каждый сервер сети и снабдить необходимыми описаниями системы. Данные описания это код на декларативном языке программирования, который использует *Cfengine*. Клиенту *cfengine* не необходимо постоянное соединение с сервером, с заданным интервалом он перечитывает файл *promises.cf* и сравнивает описанное в нем состояние системы с реальным. Если реальное состояние системы отличается от описанного, например, в системе может отсутствовать работающий процесс *daemon.telnet*, то выполняются действия описанные администратором системы. Функционал языка *cfengine* включает возможность проверки наличия в файлах строк заданных с помощью *regular expression* и проведения с ними операций.

Одна из частых задач администрирование серверов под управлением ОС *Linux* - это проверка наличия в */etc/hosts* файле соответствующих IP адреса и имени сервера. Код на *cfengine*, который бы проверил бы наличие таких строк в */etc/hosts* файле и исправил бы неправильные, выглядит следующим образом

bundle agent linux

{

files:

"/etc/hosts"

 edit\_line => linux\_hosts,

 comment => "hosts file editing";

}

bundle edit\_line linux\_hosts

{

delete\_lines:

 "^ \*$(sys.ipv4[eth0]).\*";

 "^ \*127.0.0.1.\*";

insert\_lines:

 "$(sys.ipv4[$(sys.interface)]) $(sys.host) $(sys.uqhost)";

 "127.0.0.1 localhost localhost.localdomain";

}

Для файла */etc/hosts* указывается метод редактирования *linux\_hosts*, который далее описывается. Данный код проверит наличие строк начинающихся со строки 127.0.0.1 и если их больше одной, то удалит все кроме заданной. Если строка в файле отсутствует, она будет добавлена. В коде также использованы несколько переменных *sys.interface*, *sys.host*, *sys.uqhost.* Они позволяют получить доступ к имени основного сетевого интерфейса, полному и короткому имени хоста. Таким образом, в файл */etc/hosts* будет добавлена строка не только для адреса 127.0.0.1, но и для адреса основного сетевого интерфейса, где будет запущен этот код. Решение задачи проверки владельца и прав доступа к файлу выглядит следующим образом

*bundle agent linux*

*{*

*files:*

*"/etc/passwd"*

 *perms => mog("644","root","root"),*

 *comment => "/etc/passwd permissions";*

*}*

Этот код позволяет проверять при запуске, что владельцем файла /etc/passwd является root, а права доступа 644 и исправлять их в случае не совпадения.

 Система управления конфигурациями Cfengine используется в таких крупных компаниях как Alcatel-Lucent, Cisco Systems, Nokia, KPMG, Facebook. Наличие в cfengine набора готового инструментария для решения часто возникающих задач администратора сети, а также универсальных методов редактирования файлов конфигурации делают его наиболее комплексной системой управления конфигурациями. Основной проблемой,

ограничивающей использование *cfengine* является сложность освоения.

 Задачи системного администратора корпоративной сети

Определение полного множества задач системного администратора корпоративной сети выходит за рамки данной работы и может являться темой для отдельного исследования. Будут рассмотрены только задачи по восстановлению работоспособности приложений и их настройке, а также настройке ОС. Автоматизация задач в работе относится к ОС *Red Hat Enterprise Linux 7.2* и работающим на нем приложениям. Многие из задач решались бы иначе в случае использования семейства ОС *Windows*.

В связи с этим задачи администратора корпоративной сети можно условно разделить на два типа:

* связанные с решением проблем в работе приложений;
* связанные с настройкой приложений и ОС;

Возникновение первого типа задач чаще всего связано с ошибками в коде приложений, а также человеческой ошибкой. В сложных системах, которые сегодня представляют сети передачи данных и работающие в них приложения, любая ошибка может стать следствием множества других. Например, ошибочная конфигурация приложения, примененная к нескольким десяткам серверов, может привести к необходимости перезапуска приложений на каждом из них. Перезапуск приложения, если оно не работает, является самым распространенным первым шагом в ходе решения проблем с ним. Такие задачи можно и нужно автоматизировать. Автоматический рестарт приложений позволяет системному администратору решать более приоритетные на данный момент времени задачи с возможностью последующего исследования причин проблем с приложением.

 Второй тип задач с настройками приложений и ОС связан с ростом числа приложений и серверов, а также изменениями в корпоративной сети передачи данных. Его можно условно разделить на задачи, связанные с:

* Первичной настройкой ОС;
* Изменениями настроек ОС;
* Установкой приложений;
* Изменениями настроек приложений.

Задачи первичной настройки ОС связаны с требованиями внутри организаций. Это могут быть требования к работающим в ОС процессам, установленному ПО, списку пользователей, сетевым настройками. Так как требования заранее известны, такие задачи можно и нужно автоматизировать.

Примером тут служит удаление лишнего ПО, установка нужного, заведение аккаунтов системных администраторов на сервере и установка сетевых настроек. Задачи по изменению настроек ОС могут быть связаны с изменениями настроек под конкретное приложение, они единичны и уникальны, поэтому не входят в состав задач подлежащих автоматизации.

Но есть также задачи по изменению настроек ОС, связанные с изменением требований в организации. Например, необходимо, чтобы на всех серверах был удален, если он есть аккаунт одного из системных администраторов. Или же изменилась структура *DNS* серверов и на всех серверах сети необходимо поменять настройки *DNS*. Такие задачи являются подмножеством подлежащих автоматизации.

Так как в корпоративных сетях передачи данных строгие требования к настройке ОС, то установка новых приложений в подобной среде часто является задокументированным набором действий. Такие задачи входят в состав задач, подлежащих автоматизации.

В работе подмножество задач системного администратора корпоративной сети, подлежащих автоматизации, включает:

* Задачи, связанные с первичной настройкой ОС;
* Задачи, связанные с массовыми изменениями настроек ОС;
* Задачи, связанные с установкой приложений и первичной их настройкой;
* Задачи, связанные с массовыми изменениями настроек приложений;
* Задачи, связанные с восстановлением работоспособности приложений;

Для решения проблемы автоматизации администрирования корпоративной сети необходимо разработать алгоритмы и их программную реализацию. (Предполагается на языке программирования *CFEngine* ). Ввиду дефицита объема статьи подробное описание алгоритмов здесь рассмотрим только для некоторых задач.

 Разработка алгоритма установки и слежения за сетевыми настройками

 Сетевые настройки, подлежащие автоматической настройке на ОС *Redhat Enterprise Linux 6 (RHEL 6)* включают:

- сетевой адрес, маску сети, широковещательный домен;

- список пользователей системы;

На ОС *RHEL 6* сетевые настройки находятся в текстовых файлах. Сетевые настройки используются системными процессами и после каждого изменения файлов настроек необходимо послать сигнал на уровне операционной системы процессам, чтобы они перечитали изменившийся файл.

Каждому сетевому интерфейсу соответствует файл */etc/sysconfig/network-scripts/ifcfg-X*, в котором указаны сетевой адрес, маска сети и широковещательный домен. Где X - это постфикс, определяющий имя сетевой конфигурации. В ОС *RHEL 6,* как правило это имя сетевого интерфейса. В этих же файлах может быть прописан сетевой шлюз, но рекомендацией компании *Red Hat* является его установка через файл */etc/sysconfig/network*.

 Блок 1 алгоритма установки и слежения за сетевыми настройками представлен на Рис. 4.



 Рис. 4.Блок 1 алгоритма установки и слежения за сетевыми настройками

Основным условием установки сетевых настроек будет наличие в системе сетевого интерфейса с именем *eth0* (1). Если в системе отсутствует этот интерфейс, то произойдет выход из блока алгоритма 1. Если же сетевой интерфейс существует, то необходима проверка, существует ли файл */etc/sysconfig/network-scripts/ifcfg-eth0* (2), если файла нет, то он будет создан с необходимой сетевой конфигурацией (3). Наличие файла приведет к проверке его содержимого (4), если хотя бы один из параметров конфигурации сети установлен неверно, то произойдет замена этого и остальных ошибочных параметров на нужные (5).

Ниже приведены 4 параметра, наличие которых необходимо в файле */etc/sysconfig/network-scripts/ifcfg-eth0* для правильной работы сети на сервере.

*DEVICE=eth0*

*IPADDR=172.16.2.222*

*NETMASK=255.255.255.0*

*ONBOOT=yes*

Параметр *DEVICE* содержит имя устройства в системе для которого будут применяться сетевые настройки. *IPADDR* содержит *IP* адрес интерфейса, а *NETMASK* его маску. Важно наличие параметра *ONBOOT* со значением *yes*, значение *no* означает, что при загрузке системы настройки не будут применены. В этом файле должен отсутствовать параметр *GATEWAY*. После изменения любого из указанных параметров, кроме *ONBOOT*, необходимо выполнить команду *service network restart* (12), которая установит в системе новые сетевые настройки.

Далее следует проверка, удалось ли изменить или создать файл сетевой конфигурации (6). Эта проверка необходима для случаев, когда файловая система, с файлами сетевой конфигурации, по причине ошибки диска стала доступна только для чтения. Если это так, то системному администратору будет отправлено сообщения об ошибке (13) и осуществится выход из блока алгоритма 1. Если сетевые настройки выше установлены, то останется проверить настройку сетевого шлюза. Для этого надо, чтобы файл */etc/sysconfig/network* существовал (7) и в нем была только одна строка вида *GATEWAY=IP* (9), где *IP* адрес сетевого шлюза. Если такой строки нет, или она задана неверно, то произойдет изменение файла */etc/sysconfig/network* (10), если файл удалось изменить (11) то останется выполнить перезапуск сетевой конфигурации командой *service network restart* (12). Если же файла нет, он будет создан с необходимыми сетевыми параметрами (8). Наличие этого файла в системе с правильными сетевыми параметрами (8) приведет к выходу из блока алгоритма 1, так как никаких изменений не требуется. Как и в случае с предыдущими конфигурациями присутствует проверка, удалось ли создать или изменить файл конфигурации */etc/sysconfig/network* (11). Если не удалось, то системному администратору будет отправлено сообщение с ошибкой (13). Если файл удачно исправлен или создан, то будет выполнено применение новой сетевой конфигурации (12) и выход из блока алгоритма 1.

 Список пользователей ОС *RHEL 6* содержится в файле */etc/passwd*, для каждого из описанных там пользователей, есть так же отдельная строка в файле */etc/shadow*. У каждого пользователя есть основная группа, описанная в */etc/group*, а так могут быть дополнительные группы, описанные там же. Все сервера сети поделены на 3 группы, но системные администраторы должны присутствовать на каждом из серверов. Аккаунты разработчиков должны удаляться со всех серверов, кроме тех, что находятся в сегменте сети 172.16.6.0/24. Если сервер находиться в сегменте сети 172.16.6.0/24 на нем необходимо проверить наличие аккаунтов разработчиков, если их нет то добавить.

 Добавление пользователя в систему осуществляется правкой */etc/passwd, /etc/shadow*, а так же созданием домашней директории пользователя. Для этого в файл */etc/passwd* добавляется строка вида

*test2:x:501:100:Test User:/home/test2:/bin/bash*

где *test2* имя пользователя, 501 его уникальный идентификатор в системе *UID*, 100 идентификатор его основной группы *GID*, *Test User* комментарий, */home/test2* путь к домашней директории, */bin/bash* командный интерпретатор. Каждый из этих столбцов необходимо проверить на правильность, если один из них установлен неверно, необходимо установить правильное значение и запустить системную команду *pwck*, которая проверит правильность данных в файлах */etc/passwd*, */etc/shadow*, */etc/group*.

Для каждого пользователя надо проверить наличие в */etc/shadow* строки вида

*test2:$6$m6SMs8Mm$AgthslI64b1Vv.bmrIL1HH85brXFu.iFb5k/U6BKuh.azDo5loyYeY8I97H7oDlVBZpzUSOzRxfnsq5grVYA5.:15625:0:99999:7:::*

Где *test2* имя пользователя, а второй столбец после двоеточия - это его пароль, к которому применена функция хеширования *SHA512* или просто хеш. Если хеш паролля пользователя не соответствует нужному, его необходимо сменить на правильный. После правки файла */etc/shadow* так же необходим запуск команды pwck.

Блок 2 алгоритма установки и слежения за сетевыми настройками представлен на Рис. 5.

 

 Рис. 5. Блок 2 алгоритма установки и слежения за сетевыми настройками

Сначала следует проверка, все ли аккаунты системных администраторов есть в файле */etc/passwd*, а так же правильно ли они прописаны (1). Если в ходе проверки выявляются ошибки, то происходит их исправление (2). В блоке 2 алгоритма слежения и установки сетевых настроек после каждой модификации файлов происходит проверка на ошибки (3),(7),(12),(16), если хотя бы один файл не удалось изменить, то системному администратору будет отправлено сообщение об ошибке (18). Если ошибок нет, происходит запуск системной утилиты *pwck* (4), которая проверяет и исправляет ошибки в файлах */etc/passwd*, */etc/shadow*, */etc/group*. Далее стартует проверка файла */etc/shadow* (5), если она выявит ошибки, файл будет исправлен (6) и произойдет новый запуск *pwck* (8). Если сервер находиться не в разработческом сегменте сети (9) то произойдет выход из блока 2 алгоритма. Если же сервер находиться в разработческом сетевом сегменте 172.16.6.0/24, то необходимо проверить, все ли аккаунты разработчиков перечислены в */etc/passwd* (10). Если какие-либо аккаунты отсутствуют или есть ошибки в их записях, то это необходимо исправить (11) с последующим запуском *pwck* (13). Далее часть алгоритма аналогична пунктам (5),(6),(7),(8). В ходе исправления файла */etc/passwd* (2),(11) дополнительно необходимо проверить наличие домашней директории пользователя и создать ее при отсутствии.

 **Литература**

1. Zamboni D. Learning CFEngine 3 ­–­­ New York: O’Reilly. 2012. – 186 p.
2. Rajneesh. Cfengine 3 Beginner’s Guide – New York: Packtpub. 2011. – 336 p.
3. Олифер В.Г. , Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – Спб.: Питер, 2016. – 943 с.