

Configuration Management

Успешная реализация DevOps невозможна без эффективного управления конфигурацией. Configuration Management (CM) обеспечивает контроль и прозрачность всего стека технологий, используемых в процессе разработки, развертывания и эксплуатации приложений.



by Илья Исаев

Что такое Configuration Management?

Configuration Management (CM) - это дисциплина в DevOps, которая отвечает за управление всеми аспектами изменений в программном обеспечении. Это включает в себя управление кодом, конфигурационными файлами, скриптами развертывания и другими ресурсами, необходимыми для создания и поддержки приложений.

CM обеспечивает контроль версий, отслеживание изменений и единообразную среду для разработки, тестирования и развертывания программного обеспечения. Это помогает командам работать слаженно и обеспечивать прозрачность в процессе доставки ПО.



Важность Configuration Management

1 Согласованность и повторяемость

Configuration Management (CM) обеспечивает согласованность и повторяемость процессов в инфраструктуре и помогает избежать ошибок, связанных с неконтролируемыми изменениями.

2 Управление рисками

CM позволяет ограничивать и контролировать изменения, тем самым снижая риски сбоев или проблем в рабочей среде.

3 Аудит и отчетность

CM предоставляет полную историю изменений, обеспечивая прозрачность и подотчетность, что критически важно для соблюдения нормативных требований.

4 Непрерывная интеграция и доставка

CM поддерживает процессы непрерывной интеграции и доставки, автоматизируя развертывание и уменьшая вероятность ошибок.

Зачем СМ

- фреймворк
- использование кода
- версионирование
- совместная работа
- поддержка
- повторяемость
- CD процесс

Push and Pull Models

Pull - быстрое обновление данных на серверах по интервалам

Push - кто-то контролирует время изменений

Configuration Drift - интервал для добавления разных инструментов

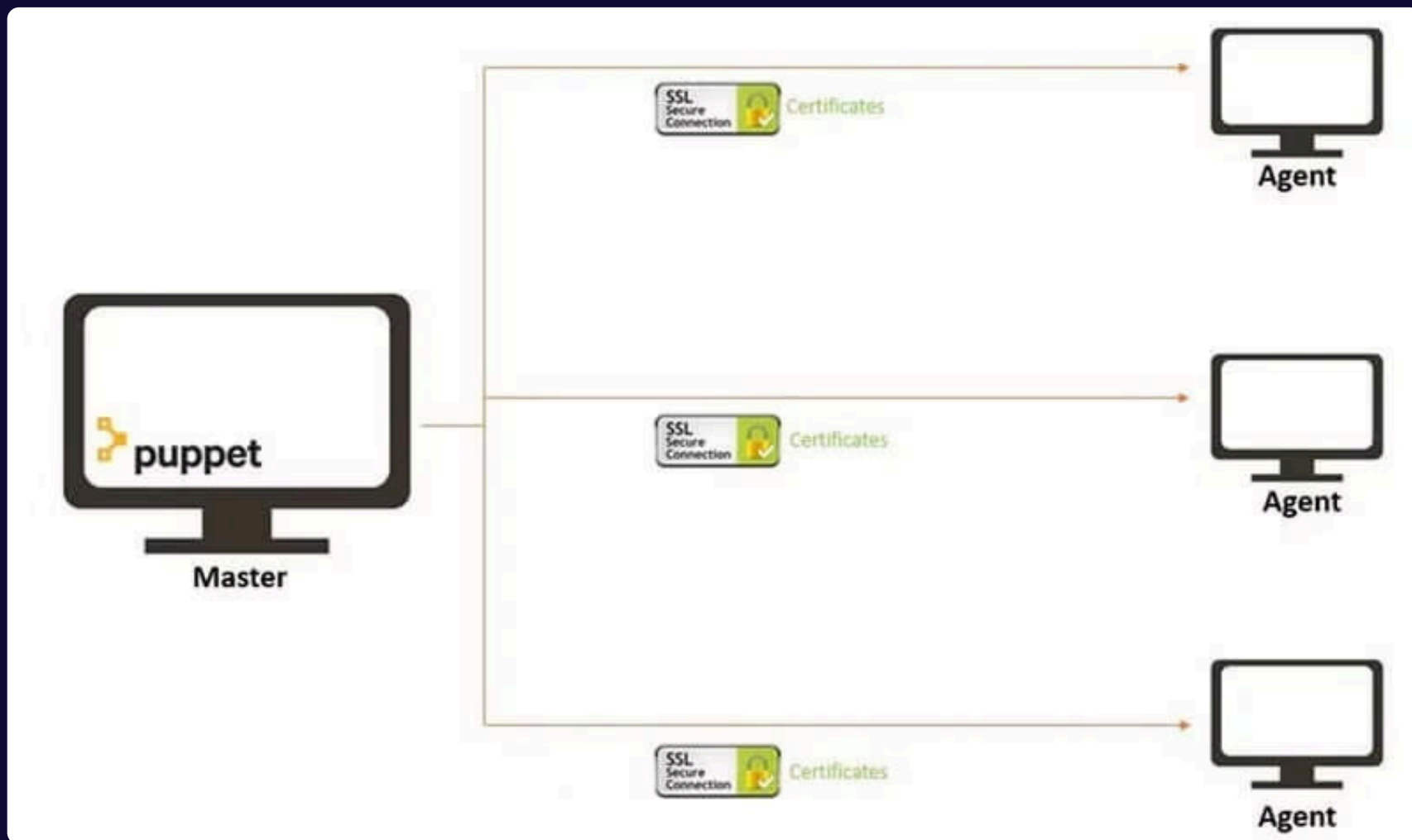
Инструменты Configuration Management

- Puppet
- Chef
- SaltStack
- Ansible

Puppet

- Клиент-серверная архитектура
- Конфигурация основанная на фактах
- Ruby
- DSL Ruby-подобный
- Коммерческая поддержка
- Сложный

Как работает



Example

```
#/etc/puppetlabs/code/environments/production/site/profile/manifests/jenkins/controller.pp
class profile::jenkins::controller (
  String $jenkins_port = '9091',
  String $java_dist    = 'jdk',
  String $java_version = 'latest',
){

  class { 'jenkins':
    configure_firewall => true,
    install_java      => false,
    port              => $jenkins_port,
    config_hash       => {
      'HTTP_PORT' => { 'value' => $jenkins_port },
      'JENKINS_PORT' => { 'value' => $jenkins_port },
    },
  }

  class { 'java':
    distribution => $java_dist,
    version      => $java_version,
    before       => Class['jenkins'],
  }
}
```



Roles and profiles example

This example demonstrates a complete roles and profiles workflow. Use it to understand the roles and profiles method as a whole. Additional examples show how to design advanced configurations...

Chef

- Клиент-серверная архитектура
- Конфигурация основана на поиске
- Ruby
- DSL Ruby
- Большое коммьюнити
- Сложно

Как работает



Example

```
if node['jenkins-server']['java']['install']
  include_recipe 'java'
end

if node['jenkins-server']['ant']['install']
  include_recipe 'ant'
end

if node['jenkins-server']['git']['install']
  include_recipe 'git'
end

if node['jenkins-server']['nginx']['install']
  include_recipe 'jenkins-server::nginx'
end

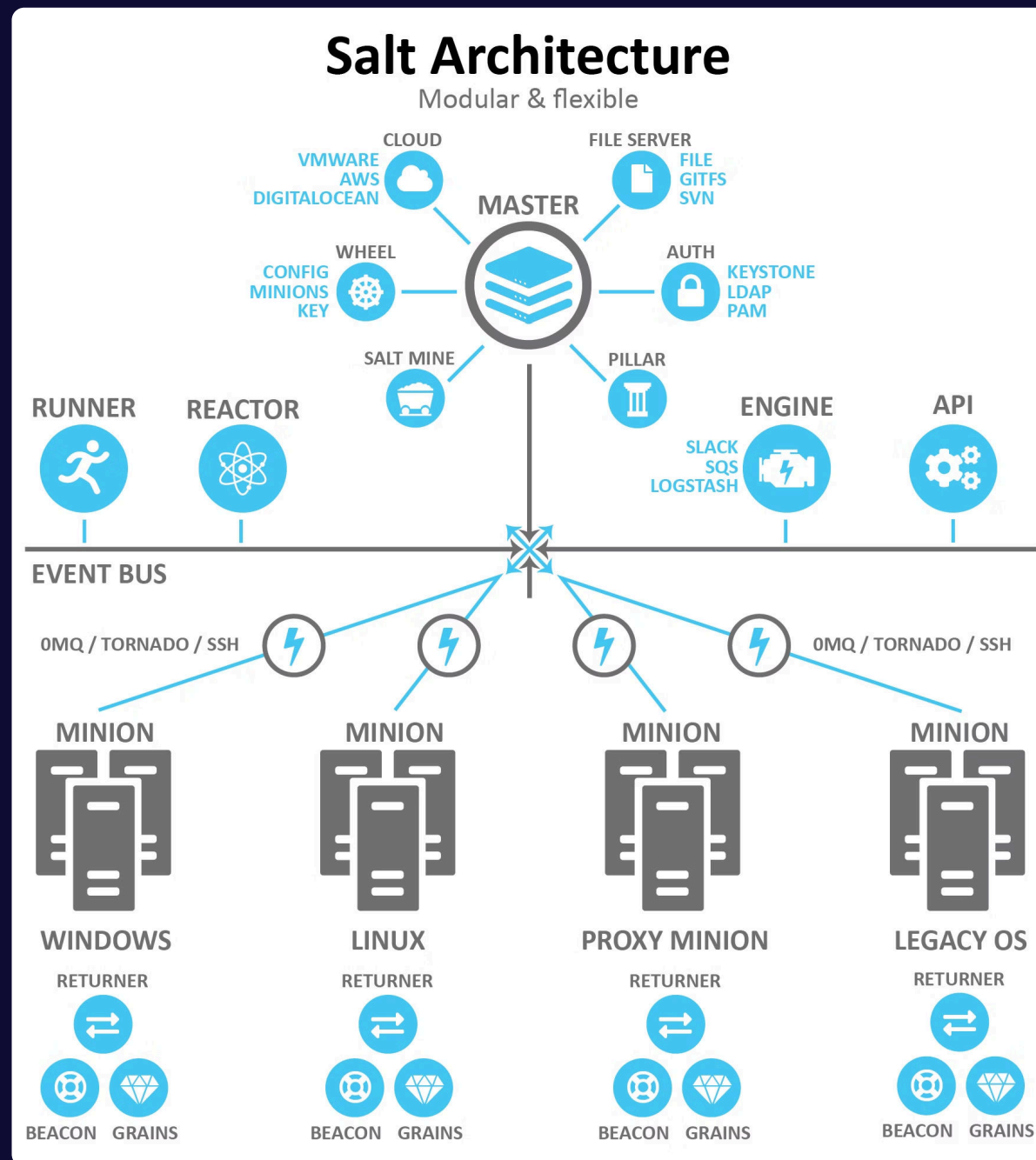
include_recipe 'jenkins-server::master'
include_recipe 'jenkins-server::settings'
include_recipe 'jenkins-server::plugins'
include_recipe 'jenkins-server::security'
include_recipe 'jenkins-server::jobs'
include_recipe 'jenkins-server::composer'

if node['jenkins-server']['slaves']['enable']
  include_recipe 'jenkins-server::slaves_credentials'
  include_recipe 'jenkins-server::slaves'
end
```

SaltStack

- Нет агента
- Клиент-серверная архитектура
- Real-time автоматизация
- Параллельное исполнение
- Python
- YAML-подобный DSL

Как работает



Example

/tmp/wildfly-8.2.1.Final.tar.gz:

file.managed:

- source: [http://olex.openlogic.com/package_versions/26491/download?](http://olex.openlogic.com/package_versions/26491/download?package_version_id=103931&path=https%3A%2F%2Folex-secure.openlogic.com%2Fcontent%2Fprivate%2F5e6a6f0815e830bba705e79e4a0470fbee8a5880%2F%2Folex-secure.openlogic.com%2Fwildfly-8.2.1.Final.tar.gz)

[package_version_id=103931&path=https%3A%2F%2Folex-](https://olex-secure.openlogic.com/content/private/5e6a6f0815e830bba705e79e4a0470fbee8a5880/olex-secure.openlogic.com/wildfly-8.2.1.Final.tar.gz)

[secure.openlogic.com%2Fcontent%2Fprivate%2F5e6a6f0815e830bba705e79e4a0470fbee8a5880%2F%2Folex-secure.openlogic.com%2Fwildfly-8.2.1.Final.tar.gz](https://olex-secure.openlogic.com/content/private/5e6a6f0815e830bba705e79e4a0470fbee8a5880/olex-secure.openlogic.com/wildfly-8.2.1.Final.tar.gz)

- template: jinja


wildfly-8.2.1.Final.tar.gz:

cmd.run:

- name: tar xf /tmp/wildfly-8.2.1.Final.tar.gz

- cwd: /opt

pratik141/
salt_linux_packages



1 Contributor 0 Issues 0 Stars 1 Fork

GitHub

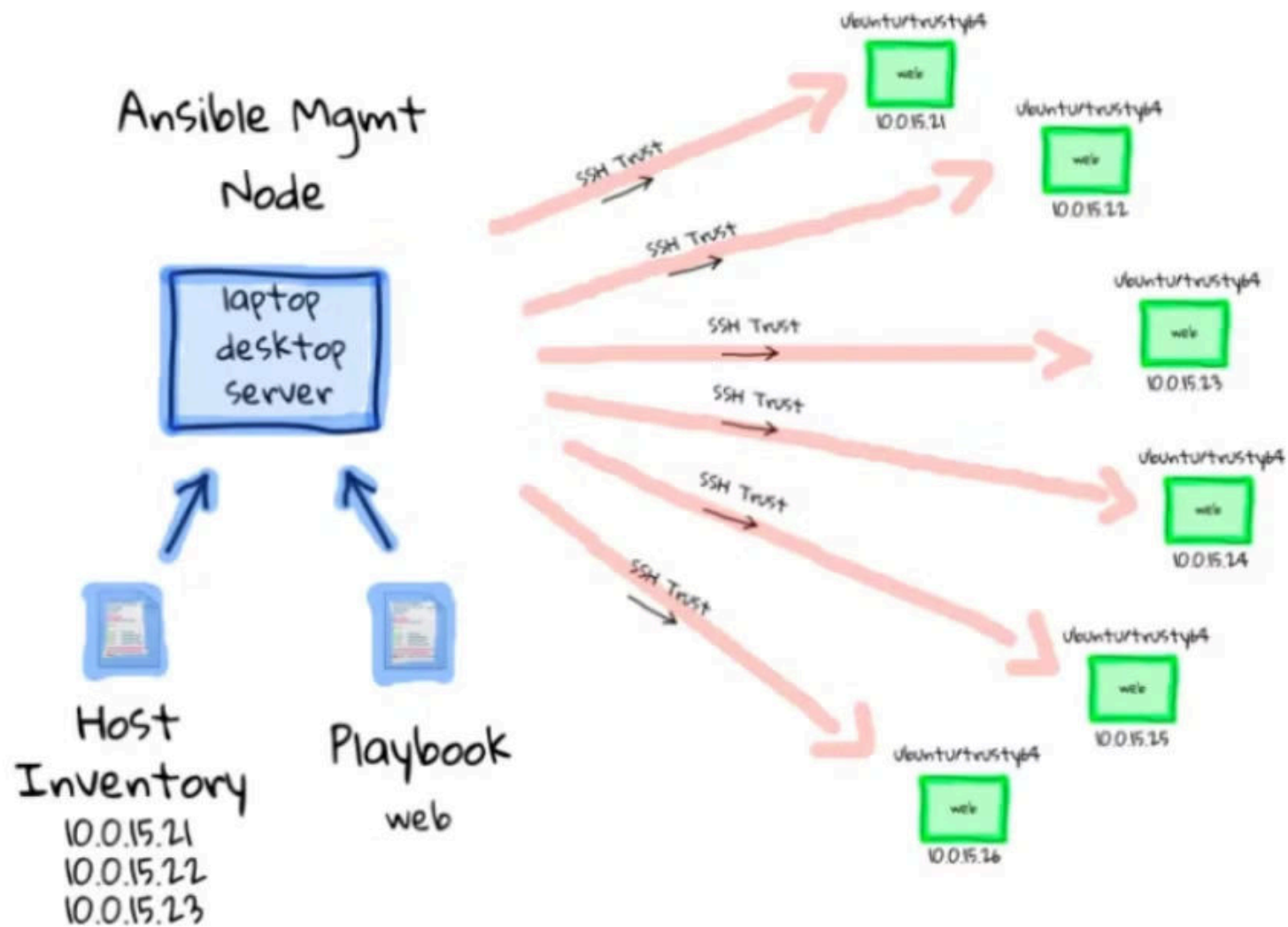
salt_linux_packages/wildfly.sls at master · pratik141...

salt_linux_packages. Contribute to pratik141/salt_linux_packages development by creating an...

Ansible

- Linux/Unix для контроля узлов
- Python
- GPL и расширяемость
- Простой
- Без агента
- Pull Model
- Идемпотентность

Как работает



Ansible Master-Slave Architecture

Example

```
---  
- name: Deploy Jenkins CI  
  hosts: jenkins_server  
  remote_user: vagrant  
  become: yes  
  
  roles:  
    - geerlingguy.repo-epel  
    - geerlingguy.jenkins  
    - geerlingguy.git  
    - tecris.maven  
    - geerlingguy.ansible  
  
- name: Deploy Nexus Server  
  hosts: nexus_server  
  remote_user: vagrant  
  become: yes  
  
  roles:  
    - geerlingguy.java  
    - savoirfairelinux.nexus3-oss  
  
- name: Deploy Sonar Server  
  hosts: sonar_server  
  remote_user: vagrant  
  become: yes  
  
  roles:  
    - wtanaka.unzip  
    - zanini.sonar  
  
- name: On Premises CentOS  
  hosts: app_server  
  remote_user: vagrant  
  become: yes  
  
  roles:  
    - jenkins-keys-config
```



 www.redhat.com



Integrating Ansible with Jenkins in a CI/CD process

The aim of this post is to demonstrate how to use Ansible for environment provisioning and application deployment in a...

- phansible

Basic Structure

```
hosts          # inventory file

group_vars/
  group1.yml    # here we assign variables to particular groups
  group2.yml
host_vars/
  hostname1.yml # here we assign variables to particular systems
  hostname2.yml

library/        # if any custom modules, put them here (optional)
module_utils/   # if any custom module_utils to support modules, put them here (optional)
filter_plugins/ # if any custom filter plugins, put them here (optional)

site.yml        # master playbook
webservers.yml  # playbook for webservers role
dbservers.yml   # playbook for dbservers role
fooapp.yml      # playbook for foo app

roles/
  common/       # this hierarchy represents defaults for a "role"
    tasks/      #
      main.yml   # <-- tasks file can include smaller files if warranted
    handlers/   #
      main.yml   # <-- handlers file
    templates/  # <-- files for use with the template resource
      ntp.conf.j2 # <----- templates end in .j2
    files/      #
      bar.txt    # <-- files for use with the copy resource
      foo.sh     # <-- script files for use with the script resource
    vars/       #
      main.yml   # <-- variables associated with this role
    defaults/   #
      main.yml   # <-- default lower priority variables for this role
    meta/       #
      main.yml   # <-- role dependencies
    library/    # roles can also include custom modules
    module_utils/ # roles can also include custom module_utils
    lookup_plugins/ # or other types of plugins, like lookup in this case

webservers/     # same kind of structure as "common" was above, done for the
webservers role
dbservers/      # ""
fooapp/         # ""
```

Basic Operators

- inventory
- plugins/libraries/modules
- tasks
- roles
- playbooks
- variables

Structure

- Control machine— управляющий хост. Сервер Ansible, с которого происходит управление другими хостами
- Manage node— управляемые хосты
- Inventory— инвентарный файл. В этом файле описываются хосты, группы хостов, а также могут быть созданы переменные
- Playbook— файл сценариев
- Play — сценарий (набор задач). Связывает задачи с хостами, для которых эти задачи надо выполнить
- Task — задача. Вызывает модуль с указанными параметрами и переменными
- Module— модуль Ansible. Реализует определенные функции

Setup

- поддержка python 3

`pip install ansible`

Group Servers

Список групп серверов для управления, два способа получения:

- Локальный файл: `/etc/ansible/hosts`
- Внешний скрипт, в официальном github-репозитории есть готовые скрипты для получения списка

Host File

`/etc/ansible/hosts` – по-умолчанию, но может быть задано переменной окружения `$ANSIBLE_HOSTS` или параметром `-i` при запуске `ansible` и `ansible-playbook`.

Пример:

```
[group1]
```

```
one.example.com
```

```
two.example.com
```

```
[group2]
```

```
three.example.com
```

Помимо списка управляемых узлов, в файле `hosts` могут быть указаны и другие сведения, необходимые для работы: номера портов для подключения по SSH, способ подключения, пароль для подключения по SSH, имя пользователя, объединения групп

Host File

Можно добавить диапазон хостов:

[routers]

192.168.255.[1:5]

Или по имени

[switches]

switch[A:D].example.com

Host File

Дочерние группы

[routers]

192.168.255.[1:5]

[switches]

switch[A:D].example.com

[devices:children]

routers

switches

Ad-hoc

```
ansible 192.168.0.1 -i hosts.ini -c network_cli -k -u user -m ios_command -a "commands='sh clock'"
```

- 192.168.0.1 – хост, к которому нужно применить действия, должен существовать в инвентарном файле
- -i myhosts.ini - параметр -i позволяет указать инвентарный файл
- -c network_cli - тип подключения. В данном случае через SSH
- -u user – выполнить от имени пользователя
- -k - аутентификация по паролю
- -m ios_command – используемый модуль
- -a "commands='sh ip int br'" – команда для выполнения

Modules

- облачные ресурсы и виртуализация (Openstack, libvirt);
- базы данных (MySQL, Postgresql, Redis, Riak);
- файлы (шаблонизация, регулярные выражения, права доступа);
- мониторинг (Nagios, monit);
- оповещения о ходе выполнения сценария (Jabber, Irc, почта, MQTT, Hipchat);
- сеть и сетевая инфраструктура (Openstack, Arista);
- управление пакетами (apt, yum, rhn-channel, npm, pacman, pip, gem);
- система (LVM, Selinux, ZFS, cron, файловые системы, сервисы, модули ядра)

Playbook

Файл с описанием действий для выполнения на хосте и группе хостов

Структура:

- play– сценарий, состоит из описания, конфигурации и списка задач
- task - конкретная задача реализуемая в рамках сценария. В задаче должно быть: – описание (название задачи можно не писать, но очень рекомендуется) – модуль и команда (действие в модуле)

```
- name: Install aptitude and required packs
hosts: workers
tasks:
  - name: Install aptitude
    apt:
      name: aptitude
      state: latest
      update_cache: true

  - name: Install required system packages
    apt:
      pkg:
        - apt-transport-https
        - ca-certificates
        - curl
        - software-properties-common
      state: latest
      update_cache: true
  - name: Add Docker GPG apt Key
    apt_key:
      url: https://download.docker.com/linux/ubuntu/gpg
      state: present
```

- Include позволяют подключать в текущий playbook файлы с задачами

Variable

- в inventory
- в playbook
- в файлах для групп/хостов
- в отдельных файлах, которые добавляются в playbook через include
- в ролях
- передавать при вызове playbook

Переменные в inventory

```
[routers]
```

```
192.168.255.[1:5]
```

```
[routers:vars]
```

```
ansible_connection=network_cli
```

```
ansible_user=user
```

```
ansible_password=password
```

Variable with playbook

```
- name: Run command on host
  hosts: workers
  gather_facts: false
  vars:
    cmd: cat /file
  tasks:
    - name: run command
      command: "{{cmd}}"
```

Variable with files groups, hosts

- Файлы переменных групп в директории “group_vars/имя_группы”;
- Файлы переменных хостов в директории “hosts_vars/имя_хоста”;
- Файлы с переменными в директории “имя_роли/vars/имя_задачи.yml”;

```
├─ group_vars _  
|   ├─ all.yml  
|   ├─ routers.yml  
|   └─ switches.yml  
├─ host_vars  
|   ├─ 192.168.0.1  
|   └─ 192.168.0.2  
└─ myhosts.ini
```


Task

```
- hosts: all
  tasks:
    - name: Print message
      debug:
        msg: Hi!
```

Roles

Роли это способ логического разбития файлов Ansible, это просто автоматизация выражений include- не нужно явно указывать полные пути к файлам с задачами или сценариями, а достаточно лишь соблюдать определенную структуру файлов

```
├── all_roles.yml
├── role1.yml
├── role2.yml
└── roles
    ├── role1
    │   ├── files
    │   ├── templates
    │   ├── tasks
    │   ├── handlers
    │   ├── vars
    │   ├── defaults
    │   └── meta
```

Roles

Все роли определены в каталоге roles

- Дочерние каталоги называются именем ролей
- Внутри каталога роли могут быть предопределенные каталоги - как минимум, tasks.
- Внутри каталогов tasks, handlers, vars, defaults, meta автоматически считывается всё, что находится в файле main.yml
- Файлы из каталогов добавляются через include, на файлы s можно ссылаться не указывая путь к ним, достаточно имени файла

Inventory

Multiple Environments (replace, merge)

```
[some_group1:children]
test_group1
test_group2
```

```
[some_group1:vars]
test_var={nginx:{port:843}}
```

```
[test_group1]
child_host1
```

```
[test_group2]
child_host2
```

```
[some_group1:children]
test_group1
test_group2
```

```
[some_group1:vars]
test_var={tomcat:{port:8888}}
```

```
[test_group1]
child_host1
```

```
[test_group2]
child_host2
```

Plugins

- callback
- filter
- action/module
- inventory
- strategia

Callback

- yaml
- debug
- slack
- logstash_callback
- ara (ansible records)

Filter

- hashi_vault (интеграция с другой системой)
- combine
- map
- select

Module/Action Plugins

- debug
- file

Inventory Plugin

- autogenerate inventory

Strategy

- mitogen (для повышения быстродействия)

Collection (Galaxy)

- nginx_core collection

Jeff Geerling

JG www.jeffgeerling.com



Jeff Geerling - Author and Software Developer in St. Louis, MO

I'm geerlingguy most places online. I'm an author, dev, and content creator in St. Louis, MO.

Other

- Jump Host/VPN
- Molecule/Testinfra

Lifecycle Molecule

- └── default
 - └── dependency
 - └── cleanup
 - └── destroy
 - └── syntax
 - └── create
 - └── prepare
 - └── converge
 - └── idempotence
 - └── side_effect
 - └── verify
 - └── cleanup
- └── destroy

- AWX (GUI)
- Ansible Builder (упаковка согласованных данных в Docker)

Заключение и ключевые выводы

1. Эффективное применение Configuration Management является ключевым элементом успешного DevOps-подхода.
2. **Управление конфигурацией** позволяет обеспечить стабильность, воспроизводимость и безопасность ИТ-инфраструктуры.
3. Использование автоматизированных инструментов значительно повышает эффективность и скорость процессов управления конфигурацией.

