нетология

Применение принципов IaaC в работе с виртуальными машинами

Олег
Букатчук

Олег Букатчук

Software Architect DevOps, crif.com

olegbukatchuk

План занятия

1. Понятие IaaC
2. Паттерны IaaC
3. Инструменты IaaC. Vargant
4. Инструменты IaaC. Ansible
5. Итоги
6. Домашнее задание

Понятие IaaC (Infrastructure as a Code)

Infrastructure as a Code

Подход **«Инфраструктура как код (IaaC)»**, который иногда называют «программируемой инфраструктурой», — это паттерн, по которому процесс создания/настройки инфраструктуры аналогичен процессу разработки программного обеспечения.

По сути, этот **паттерн** положил начало устранению границ между написанием приложений и созданием сред для этих приложений.

Это основа облачных вычислений и неотъемлемая часть DevOps методологии!

Основные преимущества IaaC

Ценность IaaC стоит на 3-х китах:

1. **Ускорение производства и вывода продукта на рынок**. Автоматизация IaaC значительно ускоряет процесс предоставления инфраструктуры для разработки, тестирования и масштабирования по мере необходимости.

Основные преимущества IaaC

Ценность IaaC стоит на 3-х китах:

2. **Стабильность среды, устранение дрейфа конфигураций.** Дрейф конфигурации происходит, когда произвольные изменения и обновления конфигурации приводят к несовпадению сред разработки, тестирования и развёртывания.

Основные преимущества IaaC

Ценность IaaC стоит на 3-х китах:

3. **Более быстрая и эффективная разработка.** Упрощая предоставление инфраструктуры и повышая её консистентность, IaaC ускоряет каждый этап жизненного цикла доставки ПО. Разработчики могут быстро подготовить «песочницы» и среды непрерывной интеграции / непрерывного развёртывания (CI/CD). Быстрее предоставляются тестовые среды, инфраструктура для проверки безопасности и юзабилити.

Главное преимущество применения IaaC

Идемпоте́нтность (лат. *idem* — тот же самый + *potens* — способный) — это свойство объекта или операции, при повторном выполнении которой мы получаем результат идентичный предыдущему и всем последующим выполнениям.

Термин предложил американский математик Бенджамин Пирс (англ. Benjamin Peirce) в статьях 1870-х годов.

Подходы к применению IaaC

Существует 3 подхода:

- **Декларативный**. “*Что*” мы делаем?
- **Императивный**. “*Как*” мы это делаем?
- **Интеллектуальный**. “*Почему*” мы это делаем?

Подходы к применению IaaC

Декларативный подход нацелен на то, чтобы описать, как должна выглядеть целевая конфигурация.

Императивный подход сфокусирован на том, какие внести изменения.

Интеллектуальный подход описывает, почему инфраструктура должна быть сконфигурирована именно так, как правило, это техническая документация проекта и бизнес требования.

Методы IaaC

Существует 3 метода применения IaaC:

- **Push**. В этом режиме конфигурация серверу отправляется управляющим сервером.
- **Pull.** В pull режиме целевой хост сам инициирует получение своей конфигурации. Просит выдать конфигурацию.
- **Гибридный (сочетает оба метода).** Используется при построении сложных многокомпонентных систем и целого набора инструментов управления конфигурациями.

Разница в том, кто инициирует изменение в конфигурации целевого хоста.

Сравнение популярных IaaC инструментов

Рассмотрим, по какой модели работает каждый из них:

Оркестратор	Вендор	Метод	Подход	Язык
Ansible	RedHat	Push	Declarative, Imperative	Python
Saltstack	Saltstack	Push / Pull	Declarative, Imperative	Python
Chef	Chef	Pull	Declarative, Imperative	Ruby
Puppet	Puppet	Pull	Declarative	Ruby
Terraform	HashiCorp	Push	Declarative	Golang

Паттерны IaaC (Infrastructure as a Code)

Паттерны

Па́ттерн (англ. *pattern*) — схема, действующая, как эффективный способ решения характерных задач проектирования, в частности проектирования компьютерных программ.

CI (Continuous Integration)

Непрерывная интеграция (CI) — практика разработки ПО, которая заключается в постоянном слиянии рабочих веток в общую основную ветку разработки, и выполнении частых автоматизированных сборок проекта.

Непрерывная интеграция позволяет снизить трудозатраты на выполнение рутинных задач команд разработки и сделать её более предсказуемой за счёт наиболее раннего обнаружения и устранения ошибок и противоречий.

Основным преимуществом данного паттерна является сокращение стоимости исправления дефекта, за счёт его раннего выявления.

CD (Continuous Delivery)

Непрерывная доставка (CD) — CI + CD.

Следующий после CI уровень.

Теперь новая версия не только создаётся и тестируется при каждом изменении кода, регистрируемом в репозитории, но и может быть оперативно запущена по одному нажатию кнопки развёртывания.

Позволяет выпускать изменения небольшими партиями, которые легко изменить или устранить, путём отката на предыдущую версию и последующего перезапуска процесса сборки с учётом исправления выявленных дефектов. Однако запуск развёртывания всё ещё происходит вручную — ту самую кнопку всё же надо кому-то нажать.

CD (Continuous Deployment)

Непрерывное развёртывание (CD) — CI + CD + CD.

После автоматизации релиза остаётся один ручной этап: одобрение (запуск в production, всё та же кнопка, которую кто-то должен нажать!).

Практика непрерывного развёртывания упраздняет ручные действия, не требуя непосредственного утверждения со стороны разработчика или любого другого ответственного лица.

Все изменения развёртываются автоматически. Обычно такая практика включена на Dev/Stage окружениях, но в Production по прежнему релизы происходят в ручную по причине высокого риска для бизнеса.

Резюмируя паттерны: CI + CD + CD

- **CD (Continuous Integration)** – Непрерывная интеграция
- **CD (Continuous Delivery)** – Непрерывная доставка
- **CD (Continuous Deployment)** – Непрерывное развёртывание

Важно! Как правило, под термином **CI/CD** подразумеваются только первые два термина: **интеграция и доставка**.

Непрерывная доставка **(Continuous Delivery)** отличается от непрерывного развертывания **(Continuous Deployment)** тем, что процесс развертывания в производственную среду должен быть подтвержден вручную.

Инструменты IaaC. Vagrant

Vagrant

Vagrant — это инструмент для создания и управления ВМ посредством использования принципов IaaC.

Vagrant сокращает время настройки среды разработки и делает оправдание разработчиков «на моём компьютере всё работает» пережитком прошлого.

Преимущества:

- **Скорость** – быстрый старт виртуального окружения;
- **Простота** – декларативный метод описания конфигураций;
- **Расширяемость** — лёгкое подключение кастомных провайдеров.

Vagrant: провайдеры

Провайдеры из коробки:

- **VirtualBox,**
- **Hyper-V,**
- **Docker.**

Прежде чем вы сможете использовать другого провайдера, вы должны установить его. Установка провайдеров осуществляется через систему плагинов Vagrant.

```
# Установка провайдера VMWare с помощью системы плагинов
$ vagrant plugin install vagrant-vmware-desktop

# Обновление провайдера VMWare с помощью системы плагинов
$ vagrant plugin update vagrant-vmware-desktop
```

Vagrant: первые шаги


```
# Проверяем установленную версию
$ vagrant --version
Vagrant 2.2.9

# Задаём провайдер по умолчанию
$ export VAGRANT_DEFAULT_PROVIDER=virtualbox
# Загружаем образ ВМ hashicorp/bionic64 для провайдера virtualbox
$ vagrant box add bento/ubuntu-20.04 --provider=virtualbox --force
==> box: Loading metadata for box 'bento/ubuntu-20.04'
    box: URL: https://vagrantcloud.com/bento/ubuntu-20.04
==> box: Adding box 'bento/ubuntu-20.04' (v202107.28.0) for provider:
virtualbox
    box: Downloading:
https://vagrantcloud.com/bento/boxes/ubuntu-20.04/versions/202107.28.0/
providers/virtualbox.box
Download redirected to host:
vagrantcloud-files-production.s3-accelerate.amazonaws.com
==> box: Successfully added box 'bento/ubuntu-20.04' (v202107.28.0) for
'virtualbox'!
# Проверяем доступные нам образы операционных систем
$ vagrant box list
bento/ubuntu-16.04 (virtualbox, 201802.02.0)
debian/stretch64   (virtualbox, 9.4.0)
bento/ubuntu-20.04
```

Структура Vagrantfile: часть 1


```
ISO = "bento/ubuntu-20.04"
NET = "192.168.192."
DOMAIN = ".netology"
HOST_PREFIX = "server"
INVENTORY_PATH = "../ansible/inventory"

servers = [
  {
    :hostname => HOST_PREFIX + "1" + DOMAIN,
    :ip => NET + "11",
    :ssh_host => "20011",
    :ssh_vm => "22",
    :ram => 1024,
    :core => 1
  }
]
```

Структура Vagrantfile: часть 2


```
Vagrant.configure(2) do |config|
  config.vm.synced_folder ".", "/vagrant", disabled: false
  servers.each do |machine|
    config.vm.define machine[:hostname] do |node|
      node.vm.box = ISO
      node.vm.hostname = machine[:hostname]
      node.vm.network "private_network", ip: machine[:ip]
      node.vm.network :forwarded_port, guest: machine[:ssh_vm],
host: machine[:ssh_host]
      node.vm.provider "virtualbox" do |vb|
        vb.customize ["modifyvm", :id, "--memory", machine[:ram]]
        vb.customize ["modifyvm", :id, "--cpus", machine[:core]]
        vb.name = machine[:hostname]
      end
    end
  end
end
```

Vagrant: первые шаги


```
# Запуск ВМ. В директории, где находится находится Vagrantfile
$ vagrant up
Bringing machine 'server1.netology' up with 'virtualbox' provider...
==> server1.netology: Checking if box 'debian/stretch64' version '9.4.0' is up to date...
==> server1.netology: Clearing any previously set network interfaces...
==> server1.netology: Preparing network interfaces based on configuration...
    server1.netology: Adapter 1: nat
    server1.netology: Adapter 2: hostonly
==> server1.netology: Forwarding ports...
    server1.netology: 22 (guest) => 20011 (host) (adapter 1)
    server1.netology: 22 (guest) => 2222 (host) (adapter 1)
==> server1.netology: Running 'pre-boot' VM customizations...
==> server1.netology: Booting VM...
==> server1.netology: Waiting for machine to boot. This may take a few minutes...
    server1.netology: SSH address: 127.0.0.1:2222
    server1.netology: SSH username: vagrant
    server1.netology: SSH auth method: private key
    server1.netology:
    server1.netology: Vagrant insecure key detected. Vagrant will automatically replace
    server1.netology: this with a newly generated keypair for better security.
    server1.netology:
    server1.netology: Inserting generated public key within guest...
    server1.netology: Removing insecure key from the guest if it's present...
    server1.netology: Key inserted! Disconnecting and reconnecting using new SSH key...
==> server1.netology: Machine booted and ready!
==> server1.netology: Checking for guest additions in VM...
    server1.netology: No guest additions were detected on the base box for this VM! Guest
    server1.netology: additions are required for forwarded ports, shared folders, host only
    server1.netology: networking, and more. If SSH fails on this machine, please install
    server1.netology: the guest additions and repackage the box to continue.
    server1.netology:
    server1.netology: This is not an error message; everything may continue to work properly,
    server1.netology: in which case you may ignore this message.
==> server1.netology: Setting hostname...
==> server1.netology: Configuring and enabling network interfaces...

==> server1.netology: Machine 'server1.netology' has a post `vagrant up` message. This is a message
==> server1.netology: from the creator of the Vagrantfile, and not from Vagrant itself:
==> server1.netology:
==> server1.netology: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports
```

Vagrant: первые шаги


```
# Заходим в ВМ. В директории, где находится находится Vagrantfile
$ vagrant ssh
vagrant@server1:~$ cat /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.2 LTS"
NAME="Ubuntu"
VERSION="20.04.2 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.2 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
vagrant@server1:~$ ip a | grep inet | grep 192
    inet 192.168.192.11/24 brd 192.168.192.255 scope global eth1
vagrant@server1:~$ hostname -f
server1.netology
vagrant@server1:~$ free
              total        used        free      shared  buff/cache   available
Mem:        1020368       31564      832932        2932      155872      852252
vagrant@server1:~$ exit
logout
Connection to 127.0.0.1 closed.
```

Vagrant: первые шаги


```
# Выключение ВМ. В директории, где находится находится Vagrantfile
$ vagrant halt
==> server1.netology: Attempting graceful shutdown of VM...

# Проверяем состояние ВМ. В директории, где находится находится Vagrantfile
$ vagrant status
Current machine states:

server1.netology          poweroff (virtualbox)

The VM is powered off. To restart the VM, simply run `vagrant up`


# Удаляем ВМ. В директории, где находится находится Vagrantfile
$ vagrant destroy
server1.netology: Are you sure you want to destroy the 'server1.netology' VM?
[y/N] y
==> server1.netology: Destroying VM and associated drives...

# Снова проверяем состояние ВМ. В директории, где находится находится Vagrantfile
$ vagrant status
Current machine states:

server1.netology          not created (virtualbox)
```

Инструменты IaaC. Ansible

Ansible

Ansible — это инструмент для управления конфигурациями.

Главное его отличие от других подобных систем в том, что Ansible использует существующую SSH инфраструктуру, в то время как другие (Saltstack, Chef, Puppet, и пр.) требуют установки специального PKI-окружения.

Преимущества:

- **Скорость** – быстрый старт на текущей SSH инфраструктуре.
- **Простота** – декларативный метод описания конфигураций.
- **Расширяемость** — лёгкое подключение кастомных ролей и модулей.

Ansible: жизненный цикл

Ansible может применятся на всех стадиях жизненного цикла инфраструктуры ваших проектов:

- **Provision**
- **Configure**
- **Deploy**
- **Operate**

Ansible поставляется с огромным количеством готовых к использованию модулей.

```
# Вывод версии Ansible
$ ansible --version
ansible 2.9.11
  config file = /Users/olegbukatchuk/git/netology.ru/virt-homeworks/05-virt-02-iaac/src/ansible/ansible.cfg
  configured module search path = ['/Users/olegbukatchuk/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/Cellar/ansible/2.9.11/libexec/lib/python3.8/site-packages/ansible
  executable location = /usr/local/bin/ansible
  python version = 3.8.5 (default, Jul 31 2020, 14:19:14) [Clang 11.0.3 (clang-1103.0.32.62)]
# С этого момента команда ниже это ваш лучший друг, не считая google.com :-)
$ ansible -h
```

Ansible: playbook (подготовка)

Определим inventory файл для Ansible playbook в котором содержится информация о ВМ:

```
[nodes:children]
manager

[manager]
server1.netology ansible_host=127.0.0.1 ansible_port=20011 ansible_user=vagrant
```

Определим файл ansible.cfg для Ansible, который содержит настройки по умолчанию для Ansible:

```
[defaults]
inventory=./inventory
deprecation_warnings=False
command_warnings=False
ansible_port=22
interpreter_python=/usr/bin/python3
```

Ansible: playbook (часть 1)

Напишем небольшой Ansible playbook который будет устанавливать Docker в создаваемую нами ВМ сразу после её создания:

```
---

  - hosts: nodes
    become: yes
    become_user: root
    remote_user: vagrant

    tasks:
      - name: Create directory for ssh-keys
        file: state=directory mode=0700 dest=/root/.ssh/

      - name: Adding rsa-key in /root/.ssh/authorized_keys
        copy: src=~/.ssh/id_rsa.pub dest=/root/.ssh/authorized_keys owner=root
mode=0600
        ignore_errors: yes

      - name: Checking DNS
        command: host -t A google.com
```

Ansible: playbook (часть 2)


```
    - name: Installing tools
      apt: >
        package={{ item }}
        state=present
        update_cache=yes
      with_items:
        - git
        - curl

    - name: Installing docker
      shell: curl -fsSL get.docker.com -o get-docker.sh && chmod +x
get-docker.sh && ./get-docker.sh

    - name: Add the current user to docker group
      user: name=vagrant append=yes groups=docker
```

Ansible: provision

Подключаем Ansible playbook к нашей Vagrant конфигурации:

```
INVENTORY_PATH = "../ansible/inventory"


node.vm.provision "ansible" do |setup|
 setup.inventory_path = INVENTORY_PATH
 setup.playbook = "../ansible/provision.yml"
 setup.become = true
 setup.extra_vars = { ansible_user: 'vagrant' }
end
```

На этом теоретическая часть закончена, переходим к практике реализации IaaC!

Итоги

Что мы узнали?

- Рассмотрели, что такое IaaC;
- Узнали о преимуществах, методах и паттернах IaaC;
- Сравнили возможности популярных IaaC инструментов;
- Научились создавать виртуальные машины применяя IaaC подход;
- Научились выполнять пост-установочную подготовку окружения внутри виртуальной машины;
- Поняли, как можно объединять Vagrant и Ansible для быстрой подготовки рабочего окружения.

Домашнее задание

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Олег Букатчук

olegbukatchuk