

Оркестрация группой Docker контейнеров на примере Docker Compose



Олег
Букатчук



Олег Букатчук

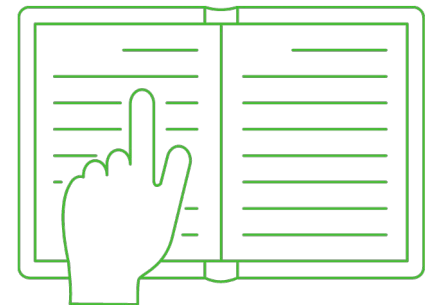
Software Architect DevOps, crif.com



Предисловие

На этом занятии мы:

- поговорим об использовании Docker Compose;
- соберем образы ОС с помощью Packer для использования их в Яндекс.Облаке;
- создадим ВМ в облаке с помощью Terraform используя свой собственный образ ОС;
- реализуем полный цикл создания Production сервиса на примере Prometheus, Grafana, Alert manager, Push gateway, Node exporter, cAdvisor, Caddy.





План занятия

1. [Введение в Docker Compose](#)
2. [Структура Docker Compose](#)
3. [Базовые команды Docker Compose](#)
4. [Вводная часть про Облака: Packer, Terraform](#)
5. [Развёртывание стека микросервисов](#)
6. [Packer + Terraform + Ansible + Docker](#)
7. [Итоги](#)
8. [Домашнее задание](#)



Введение в Docker Compose

Docker Compose

Docker Compose — это CLI утилита, дополняющая (расширяющая) функциональность Docker Engine.

Docker Compose предназначен для решения задач, связанных с развёртыванием проектов **состоящих из двух и более компонентов** (контейнеров).

Реальные проекты обычно включают в себя целый набор совместно работающих микросервисов.

Большим преимуществом использования Docker Compose является то, что можно определить стек приложения в одном файле и управлять набором микросервисов, как единым набором сущностей!

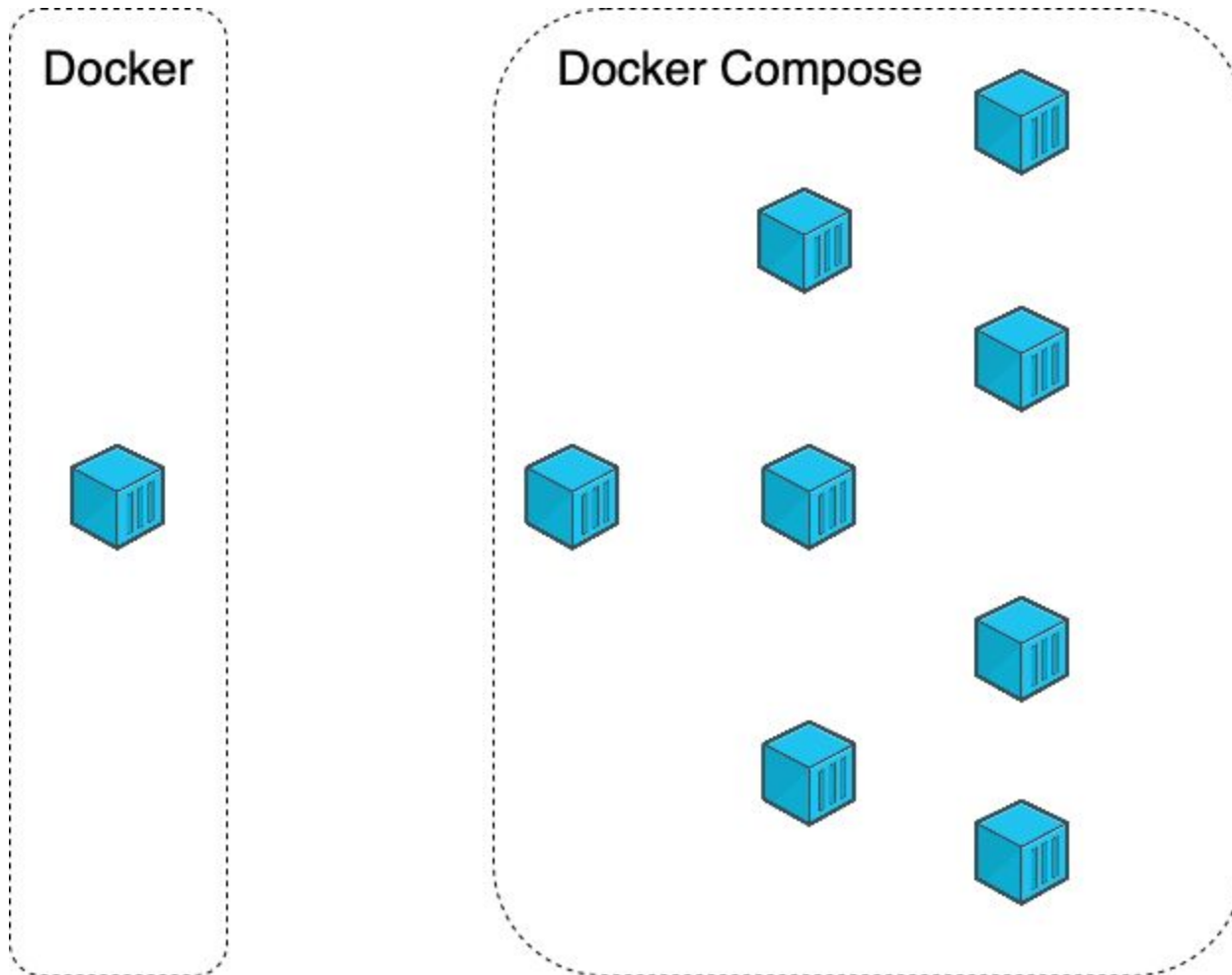
Разница между Docker и Docker Compose

Docker применяется для управления **каждым отдельным контейнером** (сервисом), из которых состоит приложение.

Docker Compose используется для одновременного управления **сразу несколькими контейнерами**, входящими в состав приложения.

Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными конфигурациями.

Разница между Docker и Docker Compose



Разница между Docker и Docker Compose

Docker



Ansible

Docker Compose



Разница между Docker и Docker Compose

Docker



Ansible

Docker Compose



Push Gateway



Karma



Alert Manager



Caddy



Prometheus



Node Exporter

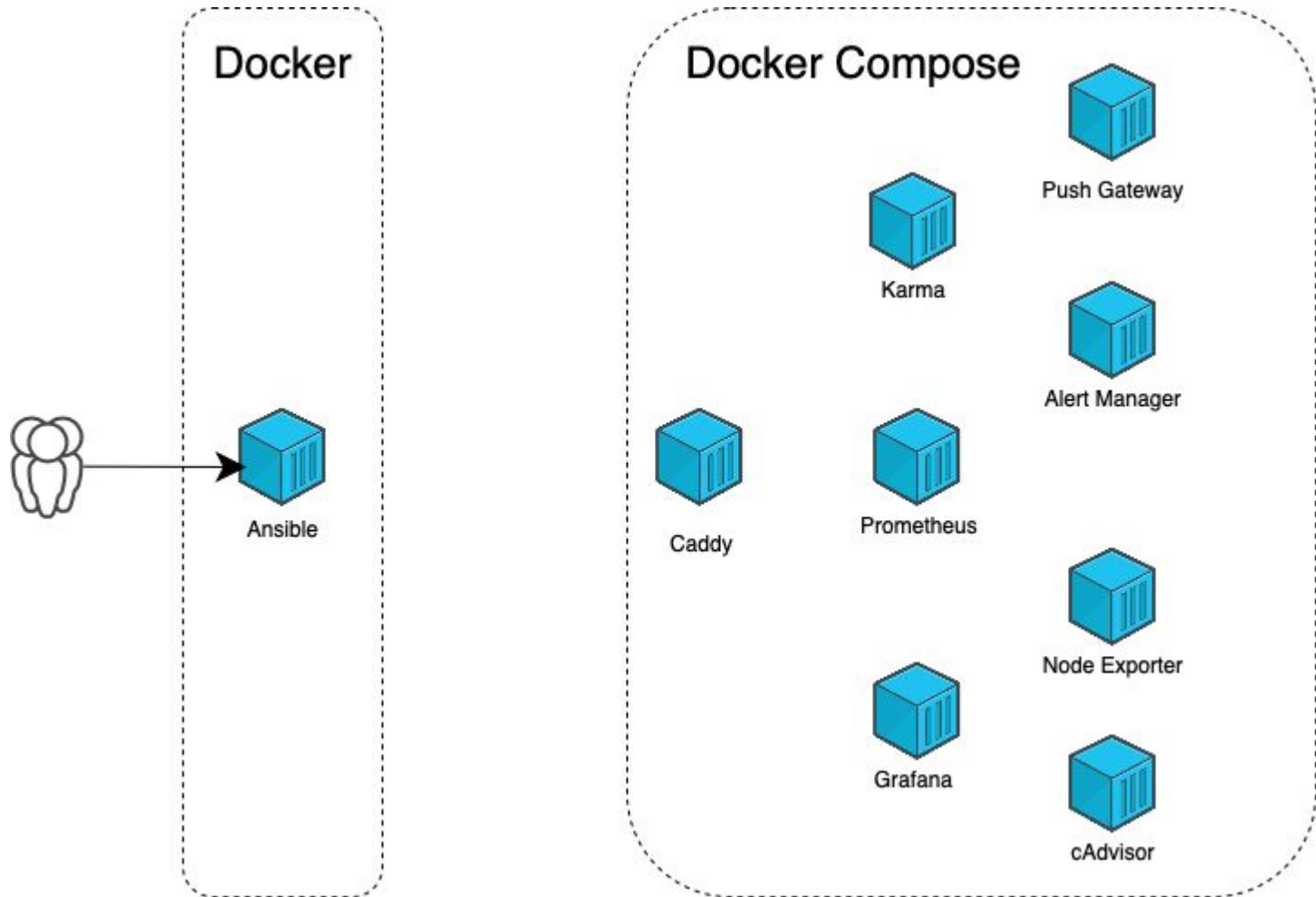


Grafana

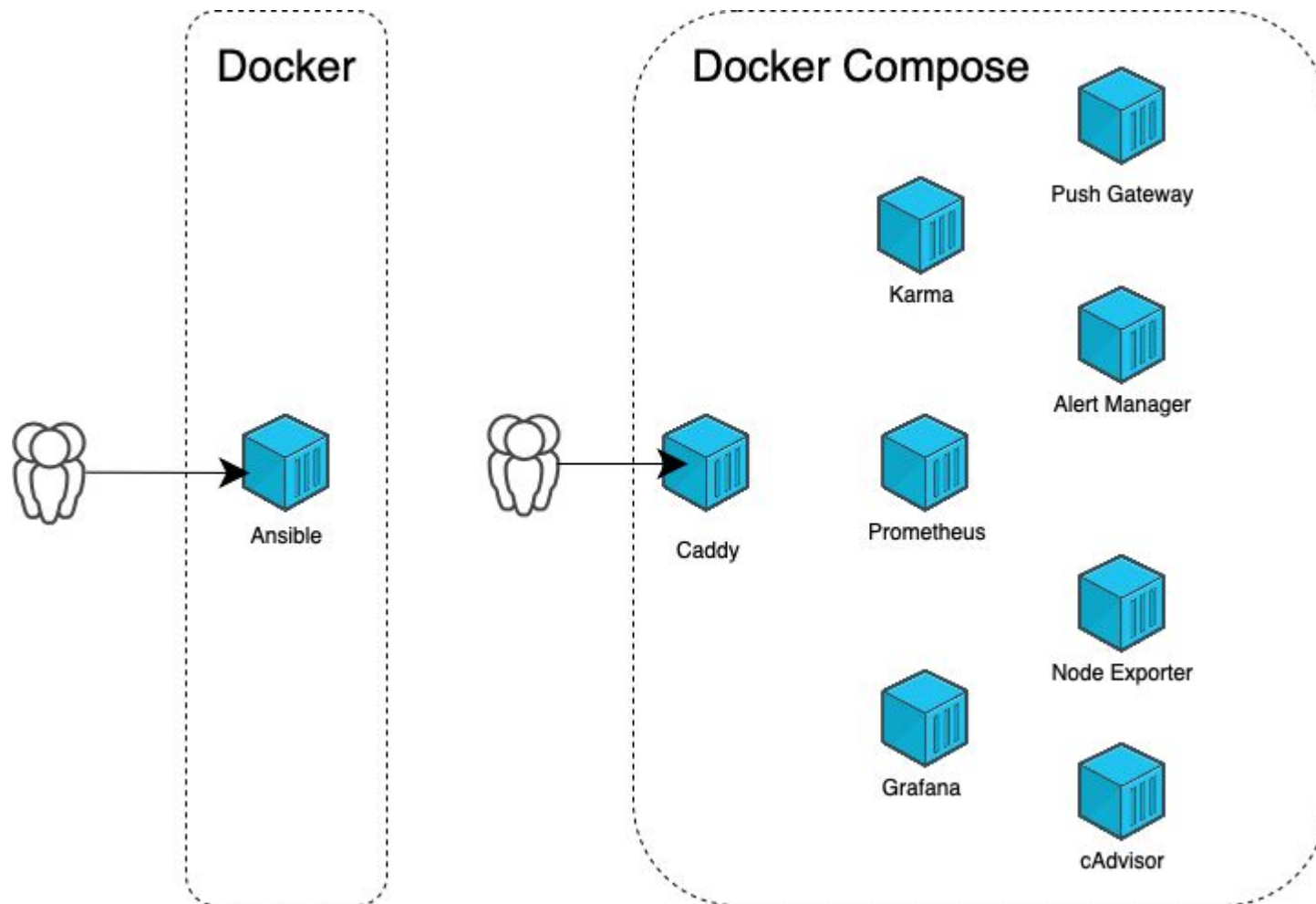


cAdvisor

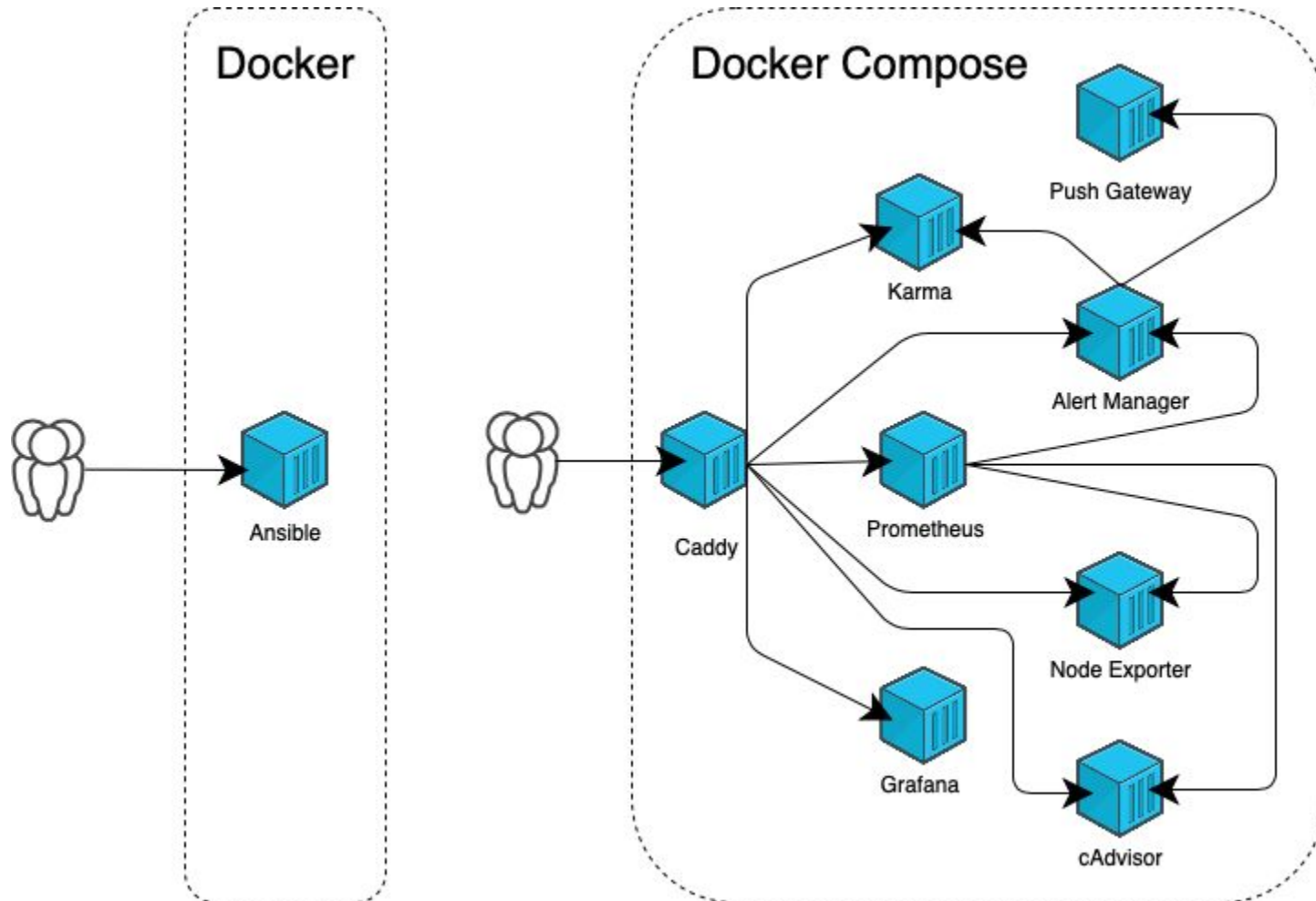
Разница между Docker и Docker Compose



Разница между Docker и Docker Compose



Разница между Docker и Docker Compose





Структура Docker Compose

Структура Docker Compose файла

- **docker-compose.yaml / docker-compose.yml**

Расширение файла может быть **.yaml** или **.yml**

```
# Структура docker-compose файла
version: '2.1'
networks:
  monitoring:
    driver: bridge
volumes:
  prometheus_data: {}
services:
  prometheus:
    image: prom/prometheus:v2.17.1
    container_name: prometheus
    volumes:
      - prometheus_data:/prometheus
    restart: always
    networks:
      - monitoring
```

Как установить Docker Compose

- Актуальную версию можно найти по [ссылке](#):

```
# Установка docker-compose
curl -L
https://github.com/docker/compose/releases/download/1.29.2/docker-compose
-`uname -s`-`uname -m` -o /usr/bin/docker-compose \
&& chmod +x /usr/bin/docker-compose
```




Базовые команды Docker Compose

Базовые команды Docker Compose

- **docker-compose build** — сборка образа.

Собранные образы помечаются, как **project_service**.

Если в Docker Compose файле указано имя образа, Docker образ помечается этим именем.

Если после первоначальной сборки вы изменяете Dockerfile, службы или содержимое ее каталога сборки (контекста сборки), **запустите docker-compose build повторно**, чтобы пересобрать изменённый docker образ.

https://docs.docker.com/engine/reference/commandline/compose_build/

Базовые команды Docker Compose

- **docker-compose pull** скачивает образы из удалённого реестра в локальный Docker Registry.

Извлекает указанный в **stanza service**: образ из удалённого реестра, определенный в файле docker-compose.yaml, но не запускает контейнер на основе этого образа, а просто помещает их в локальный реестр Docker.

https://docs.docker.com/engine/reference/commandline/compose_pull/

Базовые команды Docker Compose

- **docker-compose push** загружает образ из локально реестра в удалённый реестр (Docker Registry).

Загружает указанный в **stanza** *service*: образ в удалённый реестр, определенный в файле `docker-compose.yml`, шаблон загрузки ищет по имени образа `registry/repository/image:tag`

<https://docs.docker.com/compose/reference/push/>

Базовые команды Docker Compose

- **docker-compose up (-d)** извлекает/собирает/запускает контейнеры на основе указанных в **stanza service**: образов.

```
docker-compose up -d (--detach)
```

Команда запускает контейнеры в фоновом режиме и оставляет их работающими.

Если процесс обнаруживает ошибку, код выхода для этой команды — **1**. Если процесс прерывается с помощью **SIGINT** (ctrl + C) или **SIGTERM**, контейнеры останавливаются, а код выхода равен — **0**.

https://docs.docker.com/engine/reference/commandline/compose_up/

Базовые команды Docker Compose

- **docker-compose logs (-f)** выводит в STDOUT логи контейнеров на основе указанных в **stanza service**: запущенных контейнеров.

```
docker-compose logs -f (--follow)
```

Команда запускает непрерывный вывод логов в STDOUT, всех запущенных контейнеров.

```
docker-compose logs -f service_name
```

Команда запускает непрерывный вывод логов в STDOUT, **service_name** контейнера.

https://docs.docker.com/engine/reference/commandline/compose_logs/

Базовые команды Docker Compose

- **docker-compose ps** (-a) выводит список запущенных контейнеров на основе указанных в **stanza service**: в docker-compose файле.

```
docker-compose ps -a (--all)
```

Команда выводит список всех, даже остановленных контейнеров на основе указанных в **stanza service**: в docker-compose файле.

<https://docs.docker.com/compose/reference/ps/>

Базовые команды Docker Compose

- **`docker-compose top`** выводит список запущенных процессов внутри контейнеров на основе указанных в **`stanza service`**: в `docker-compose` файле.

Удобно использовать при отладке взаимодействия нескольких контейнеров. Позволяет посмотреть от какого пользователя запущен процесс, его PID, а также потребление CPU каждого контейнера, но лучше использовать утилиту `ctop`.

```
# Установка утилиты ctop
$ curl -L
https://github.com/bcicen/ctop/releases/download/0.7.6/ctop-0.7.6-li
nux-amd64 -O /usr/bin/ctop && chmod +x /usr/bin/ctop
```

<https://docs.docker.com/compose/reference/ps/>

Базовые команды Docker Compose


- **`docker-compose down`** останавливает все запущенные контейнеры на основе указанных в **`stanzas`** *service*: в `docker-compose` файле.

Удаляет все контейнеры, сети, тома и образы, созданные с помощью `up` (-d). По умолчанию удаляются только следующие элементы:

- **Контейнеры для сервисов**, определенных в Compose файле.
- **Сети**, определенные в разделе сетей Compose файла.
- **Сеть по умолчанию**, если таковая используется.

Важно: Сети и тома, определенные как внешние, никогда не удаляются!

https://docs.docker.com/engine/reference/commandline/compose_down/



Вводная часть про Облака: Packer, Terraform

Packer



Packer — это инструмент для создания одинаковых образов ОС для различных платформ из одного описания.

Он отлично дружит и со всеми крупными облачными провайдерами, вроде **AWS**, **GCE**, **Azure** и **Digital Ocean**, и даже с локальными гипервизорами, вроде **VMWare** и **VirtualBox**.

Создавать образы можно как для Linux, так и для Windows.

<https://www.packer.io/>

Terraform




Terraform — это инструмент помогающий декларативно управлять инфраструктурой.

Используя Terraform не приходится (в консоли вашего облачного провайдера) вручную создавать диски, инстансы, учётные записи, сети и т.д.

Так реализуется важнейший IaaS принцип: инфраструктура хранится в системе контроля версий точно так же, как исходный код, следовательно её можно рецензировать или откатывать к более раннему состоянию.

<https://www.terraform.io/>



Развертывание стека микросервисов

Авторизация в Yandex.Cloud

Для работы с Яндекс Облаком вам **потребуется установить**
[утилиту yc](#).

```
# Актуальная версия для macOS (на момент подготовки презентации).  
$ yc --version  
Yandex.Cloud CLI 0.82.0 darwin/amd64  
  
# Справка по командам утилиты yc  
$ yc --help
```

<https://cloud.yandex.ru/docs/cli/quickstart>

Авторизация в Yandex.Cloud

```
# Инициализация профиля
$ yc init
Welcome! This command will take you through the configuration
process.
Pick desired action:
  [1] Re-initialize this profile 'yc_recovery' with new settings
  [2] Create a new profile
Please enter your numeric choice: 2
Enter profile name. Names start with a lower case letter and contain
only lower case letters a-z, digits 0-9, and hyphens '-': netology
Please go to
https://oauth.yandex.ru/authorize?response_type=token&client_id=1a69
40aa636648e9b2ef845d27bec2ec in order to obtain OAuth token.
```

Авторизация в Yandex.Cloud

```
# Инициализация профиля
Please enter OAuth token: AQIAAAWX00wAATukvqUL29UBkxhnH-lGv22qUw
You have one cloud available: 'cloud-bukatchuk' (id =
b1gu1gt5nqi6lqgu3t7s). It is going to be used by default.
Please choose folder to use:
[1] default (id = b1ghcfrf9jjd63ngp64p)
[2] netology (id = b1gaec42k169jqpo02f7)
[3] Create a new folder
Please enter your numeric choice: 2
Your current folder has been set to 'netology' (id =
b1gaec42k169jqpo02f7).
Do you want to configure a default Compute zone? [Y/n] y
Which zone do you want to use as a profile default?
[1] ru-central1-a
[2] ru-central1-b
[3] ru-central1-c
[4] Don't set default zone
Please enter your numeric choice: 1
Your profile default Compute zone has been set to 'ru-central1-a'.
```


Авторизация в Yandex.Cloud

```
# Инициализация профиля
$ yc config list
token: AQTAAAWX00wAATukvqUL29UBkxhnH-lGv22qUw
cloud-id: b1gu1gt5nqi6lqgu3t7s
folder-id: b1gaec42k169jqpo02f7
compute-default-zone: ru-central1-a

$ yc compute image list
+----+-----+-----+-----+-----+
| ID | NAME | FAMILY | PRODUCT IDS | STATUS |
+----+-----+-----+-----+-----+
+----+-----+-----+-----+-----+
```

Создание сети в Yandex.Cloud

```
# Инициализация сети
$ yc vpc network create \
>     --name net \
>     --labels my-label=netology \
>     --description "my first network via yc"
```

```
id: enp6o83r23jge62evv45
folder_id: b1gaec42k169jqpo02f7
created_at: "2021-10-02T13:01:01Z"
name: net
description: my first network via yc
labels:
  my-label: netology
```

Создание подсети в Yandex.Cloud

```
# Инициализация подсети
$ yc vpc subnet create \
>   --name my-subnet-a \
>   --zone ru-central1-a \
>   --range 10.1.2.0/24 \
>   --network-name net \
>   --description "my first subnet via yc"
id: e9bnppf2hf7326hqag94
folder_id: b1gaec42k169jqpo02f7
created_at: "2021-10-02T13:06:29Z"
name: my-subnet-a
description: my first subnet via yc
network_id: enp6o83r23jge62evv45
zone_id: ru-central1-a
v4_cidr_blocks:
- 10.1.2.0/24
```

Создание образа ОС в Yandex.Cloud

```
# Проверка версии Packer, корректности конфигурации и запуск сборки
$ packer --version
1.6.1
$ packer validate centos-7-base.json
$ packer build centos-7-base.json
yandex: output will be in this color.
==> yandex: Creating temporary ssh key for instance...
==> yandex: Using as source image: fd85ck0tjp72cp9jiqbi (name:
"centos-7-1612278783", family: "centos-7")
==> yandex: Use provided subnet id e9bnppf2hf7326hqag94
==> yandex: Creating instance...
==> yandex: Waiting for instance with id fhmasb12cjnmhbb4ef98 to
become active...
    yandex: Detected instance IP: 84.252.131.52
==> yandex: Using ssh communicator to connect: 84.252.131.52
==> yandex: Waiting for SSH to become available...
==> yandex: Connected to SSH!
==> yandex: Provisioning with shell script:
...
```

Создание образа ОС в Yandex.Cloud

```
# Завершение сборки
...
==> yandex: Stopping instance...
==> yandex: Deleting instance...
    yandex: Instance has been deleted!
==> yandex: Creating image: centos-7-base
==> yandex: Waiting for image to complete...
==> yandex: Destroying boot disk...
    yandex: Disk has been deleted!
Build 'yandex' finished.

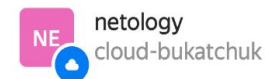
==> Builds finished. The artifacts of successful builds are:
--> yandex: A disk image was created: centos-7-base (id: fd8eam19jsb479spvg7o)
with family name centos
```

```
$ yc compute image list
```


ID	NAME	FAMILY	PRODUCT IDS	STATUS
fd8eam19jsb479spvg7o	centos-7-base	centos	f2entd2q3vii79nbabck	READY

Образ ОС доступен в UI: Yandex.Cloud

≡ Yandex Cloud




< Folder

Compute Cloud 

Service

- Virtual machines
- Disks
- Snapshots
- Images**
- Instance groups
- Placement groups
- Operations

Images

All statuses 

<input type="checkbox"/>	Name	Description	Size	Status	Date created	Optimize for disk
<input type="checkbox"/>	centos-7-base	by packer	10 GB	Ready	20 October 2021, at 11:01	no

Создание ВМ в Yandex.Cloud

```
# Проверка версии Terraform и инициализация конфигурации.  
$ terraform --version  
Terraform v1.0.8  
on darwin_amd64  
$ terraform init  
Initializing the backend...  
  
Initializing provider plugins...  
- Finding latest version of yandex-cloud/yandex...  
- Installing yandex-cloud/yandex v0.64.1...  
- Installed yandex-cloud/yandex v0.64.1 (self-signed, key ID  
E40F590B50BB8E40)  
  
Partner and community providers are signed by their developers.  
If you'd like to know more about provider signing, you can read  
about it here:  
https://www.terraform.io/docs/cli/plugins/signing.html  
  
...
```

Создание ВМ в Yandex.Cloud

```
# Проверка версии Terraform и инициализация конфигурации.
```

```
...
```

```
Terraform has created a lock file .terraform.lock.hcl to record the  
provider  
selections it made above. Include this file in your version control  
repository  
so that Terraform can guarantee to make the same selections by default  
when  
you run "terraform init" in the future.
```

```
Terraform has been successfully initialized!
```

```
You may now begin working with Terraform. Try running "terraform plan" to  
see  
any changes that are required for your infrastructure. All Terraform  
commands  
should now work.
```

```
If you ever set or change modules or backend configuration for Terraform,  
rerun this command to reinitialize your working directory. If you forget,  
other  
commands will detect it and remind you to do so if necessary.
```


Создание ВМ в Yandex.Cloud

```
# Запуск проверки плана Terraform
$ terraform plan
```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

- + create

Terraform will perform the following actions:

...

...

Plan: 3 to add, 0 to change, 0 to destroy.

Changes to Outputs:

- + external_ip_address_node01_yandex_cloud = (known after apply)
- + internal_ip_address_node01_yandex_cloud = (known after apply)

Создание ВМ в Yandex.Cloud

```
# Применение Terraform плана
$ terraform apply
Plan: 3 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + external_ip_address_node01_yandex_cloud = (known after apply)
  + internal_ip_address_node01_yandex_cloud = (known after apply)
yandex_vpc_network.default: Creating...
yandex_vpc_network.default: Creation complete after 1s
[id=enpnuvnl45lmmqp2kl27]
yandex_vpc_subnet.default: Creating...
yandex_vpc_subnet.default: Creation complete after 1s
[id=e9b2nd8kum9504mul64f]
yandex_compute_instance.node01: Creating...
yandex_compute_instance.node01: Creation complete after 42s
[id=fhmg97nus7rhdceh8gjr]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
Outputs:

external_ip_address_node01_yandex_cloud = "178.154.205.74"
internal_ip_address_node01_yandex_cloud = "192.168.101.20"
```

Деплой ПО и стека микросервисов на виртуальную машину в Yandex.Cloud

```
# Подготовка ПО и запуск стека микросервисов
$ ansible-playbook provision.yml

PLAY [nodes] *****
TASK [Gathering Facts] *****
The authenticity of host '178.154.205.74 (178.154.205.74)' can't be established.
ECDSA key fingerprint is SHA256:j0bCZJ2uBZ0wXsgun7UzENtkd5H1gDJD2Kb8vAZl94A.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
ok: [node01.netology.cloud]

TASK [Create directory for ssh-keys] *****
ok: [node01.netology.cloud]

TASK [Adding rsa-key in /root/.ssh/authorized_keys] *****
changed: [node01.netology.cloud]

TASK [Checking DNS] *****
changed: [node01.netology.cloud]

TASK [Installing tools] *****
changed: [node01.netology.cloud] => (item=['git', 'curl'])
```

Деплой ПО и стека микросервисов на виртуальную машину в Yandex.Cloud

```
# Подготовка ПО и запуск стека микросервисов
TASK [Add docker repository] *****
changed: [node01.netology.cloud]

TASK [Installing docker package] *****
changed: [node01.netology.cloud] => (item=['docker-ce', 'docker-ce-cli',
'containerd.io'])

TASK [Enable docker daemon] *****
changed: [node01.netology.cloud]

TASK [Install docker-compose] *****
changed: [node01.netology.cloud]

TASK [Synchronization] *****
changed: [node01.netology.cloud]
```

Деплой ПО и стека микросервисов на виртуальную машину в Yandex.Cloud

```
# Подготовка ПО и запуск стека микросервисов

TASK [Pull all images in compose]
*****
changed: [node01.netology.cloud]

TASK [Up all services in compose]
*****
changed: [node01.netology.cloud]

PLAY RECAP
*****
*
node01.netology.cloud      : ok=12    changed=10    unreachable=0
failed=0      skipped=0    rescued=0     ignored=0
```

Теперь можно перейти по адресу:

http://внешний_ip_адрес_вашей_вм:3000

и авторизоваться в Grafana с логином/паролем (**admin/admin**)



Packer + Terraform + Ansible + Docker

Packer + Terraform + Ansible + Docker

Packer **собирает образ** на временной ВМ, которую создаёт сам для сборки образа, и затем **загружает собранный образ ВМ в S3 хранилище** этого облака.

Terraform **использует этот образ**, как отправную точку с которой начинает строить **план создания** инстанса виртуальной машины в этом облаке на основе манифеста, оформленного в виде исходного кода на языке **HCL**.

Ansible **по факту доступности виртуальной машины** начинает подготовку операционной системы созданной ВМ, его цель — подготовка окружения (установка всех зависимостей, для того чтобы запустить Docker контейнеры.)



Итоги

Итоги

Сегодня мы

- узнали о преимуществах, использования Docker Compose;
- научились собирать образы ОС с помощью Packer для использования их в Яндекс.Облаке;
- научились создавать ВМ в облаке с помощью Terraform используя свой собственный образ ОС;
- поняли, как можно использовать Docker, Docker Compose, Packer, Terraform и Ansible.
- реализовали полный цикл создания Production сервиса на примере Prometheus, Grafana, Alert manager, Push gateway, Node exporter, cAdvisor, Caddy.

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**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Олег Букатчук