Создание кластера Kubernetes с помощью Kubeadm в Ubuntu 18.04 на платформе Базис. Digital Energy

Введение

[Kubernetes](https://kubernetes.io/) — это система оркестровки контейнеров, обеспечивающая управление контейнерами в масштабе. Система Kubernetes была первоначально разработана Google на основе опыта компании в использовании контейнеров в рабочей среде. Это решение с открытым исходным кодом, и в его разработке активно участвуют представители сообщества разработчиков со всего мира.

**Примечание.** В этом обучающем модуле используется версия Kubernetes 1.24, последняя официальная поддерживаемая версия на момент публикации данной статьи. Актуальную информацию о последней версии можно найти в [текущих примечаниях к выпуску](https://kubernetes.io/docs/setup/release/notes/) в официальной документации Kubernetes.

[Kubeadm](https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm/) автоматизирует установку и настройку компонентов Kubernetes, в том числе сервера API, Controller Manager и Kube DNS. Однако данное средство не создает пользователей и не выполняет установку зависимостей уровня операционной системы и их конфигурации. Для предварительных задач существует возможность использования инструментов управления конфигурацией, таких как [Ansible](https://www.ansible.com/) и [SaltStack](https://saltstack.com/). Использование этих инструментов упрощает создание дополнительных кластеров или воссоздание существующих кластеров, а также снижает вероятность ошибок.

В этом обучающем модуле вы научитесь создавать кластер Kubernetes с помощью Ansible и Kubeadm, а затем развертывать в нем приложение Nginx в контейнерах.

Цели

Ваш кластер будет включать следующие физические ресурсы:

* **Один главный узел**

Главный узел (под *узлом* в Kubernetes подразумевается сервер), отвечающиой за управление состоянием кластера. На нем работает [система Etcd](https://github.com/coreos/etcd), которая хранит данные кластера среди компонентов, распределяющих рабочие задачи по рабочим узлам.

* **Два рабочих узла**

Рабочие узлы — это серверы, где выполняются рабочие *задачи* (т. е. приложения и службы в контейнерах). Рабочий узел продолжает выполнять назначенную задачу, даже если главный узел отключается после распределения задач. Добавление рабочих узлов позволяет увеличить объем кластера.

После прохождения данного обучающего модуля вы получите кластер, готовый к запуску приложений в контейнерах, при условии, что серверы кластера имеют достаточные ресурсы процессорной мощности и оперативной памяти для выполнения этих приложений. Практически любые традиционные приложения Unix, в том числе веб-приложения, базы данных, демоны и инструменты командной строки, можно поместить в контейнеры и запускать в кластере. Сам кластер потребляет примерно 300-500 МБ оперативной памяти и 10% ресурсов процессора на каждом узле.

После настройки кластера вы развернете веб-сервер [Nginx](https://nginx.org/en/) для проверки правильного выполнения рабочих задач.

Предварительные требования

* Пара ключей SSH на локальном компьютере под управлением Linux/macOS/BSD.
* Три сервера с Ubuntu 18.04, развернутые на технологической платформе ПАО Ростелеком Базис. Digital Energy, имеющие не менее 2 ГБ оперативной памяти и 2 виртуальных процессоров каждый. Вы должны иметь возможность подключаться к каждому серверу через SSH как пользователь root, используя вашу пару ключей SSH.
* Система Ansible, установленная на локальном компьютере.
* Знакомство с плейбуками Ansible.
* Умение запускать контейнеры из образа Docker.

Шаг 1 — Настройка каталога рабочего пространства и файла инвентаризации Ansible

В этом разделе вы создадите на локальном компьютере каталог, который будет выступать в качестве рабочего пространства. Вы выполните локальную настройку Ansible, чтобы обеспечить возможность связи с вашими удаленными серверами и выполнения команд на этих серверах. После этого вы создадите файл hosts с данными инвентаризации, в том числе с IP-адресами ваших серверов и данными групп, к которым принадлежит каждый сервер.

Из трех ваших серверов один сервер будет главным сервером, и его IP-адрес будет отображаться как master\_ip. Другие два сервера будут рабочими узлами и будут иметь IP-адреса worker\_1\_ip и worker\_2\_ip.

Создайте каталог ~/kube-cluster в домашнем каталоге локального компьютера и перейдите в него с помощью команды cd:

mkdir ~/kube-cluster

cd ~/kube-cluster

В рамках этого обучающего модуля данный каталог будет выступать в качестве рабочего пространства, и в нем будут храниться все ваши плейбуки Ansible. Также в этом каталоге вы будете запускать все локальные команды.

Создайте файл с именем ~/kube-cluster/hosts с помощью nano или своего любимого текстового редактора:

nano ~/kube-cluster/hosts

Добавьте в файл следующий текст с информацией о логической структуре вашего кластера:

[masters]

master ansible\_host=master\_ip ansible\_user=root

[workers]

worker1 ansible\_host=worker\_1\_ip ansible\_user=root

worker2 ansible\_host=worker\_2\_ip ansible\_user=root

[all:vars]

ansible\_python\_interpreter=/usr/bin/python3

Возможно вы помните, что [*файлы инвентаризации*](http://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html) в Ansible используются для указания данных серверов, в том числе IP-адресов, удаленных пользователей и группировок серверов как единый объем для целей выполнения команд. Файл ~/kube-cluster/hosts будет вашим файлом инвентаризации, и вы добавили в него две группы Ansible (**masters** и **workers**) для определения логической структуры вашего кластера.

В группе **masters** имеется запись сервера master, в которой указан IP-адрес главного узла (master\_ip) и указывается, что система Ansible должна запускать удаленные команды от имени пользователя root.

В группе **workers** также есть две записи для серверов рабочих узлов (worker\_1\_ip и worker\_2\_ip), где пользователь ansible\_user также задается как пользователь root.

В последней строке файла Ansible предписывается использовать для операций управления интерпретаторы Python 3 удаленных серверов.

Сохраните и закройте файл после добавления текста.

После настойки инвентаризации сервера с помощью групп мы переходим к установке зависимостей уровня операционной системы и создания параметров конфигурации.

Шаг 2 — Создание пользователя без привилегий root на всех удаленных серверах

В этом разделе вы создадите пользователя без привилегий root с привилегиями sudo на всех серверах, чтобы вы могли вручную подключаться к ним через SSH как пользователь без привилегий. Это полезно на случай, если вы захотите посмотреть информацию о системе с помощью таких команд как top/htop, просмотреть список работающих контейнеров или изменить файлы конфигурации, принадлежащие пользователю root. Данные операции обычно выполняются во время технического обслуживания кластера, и использование пользователя без привилегий root для выполнения таких задач минимизирует риск изменения или удаления важных файлов или случайного выполнения других опасных операций.

Создайте в рабочем пространстве файл с именем ~/kube-cluster/initial.yml:

nano ~/kube-cluster/initial.yml

Добавьте в файл следующую строку сценария play для создания пользователя без привилегий root с привилегиями sudo на всех серверах. Сценарий в Ansible — это набор выполняемых шагов, нацеленных на определенные серверы и группы. Следующий сценарий создаст пользователя без привилегий root с привилегиями sudo:

- hosts: all

become: yes

tasks:

- name: create the 'ubuntu' user

user: name=ubuntu append=yes state=present createhome=yes shell=/bin/bash

- name: allow 'ubuntu' to have passwordless sudo

lineinfile:

dest: /etc/sudoers

line: 'ubuntu ALL=(ALL) NOPASSWD: ALL'

validate: 'visudo -cf %s'

- name: set up authorized keys for the ubuntu user

authorized\_key: user=ubuntu key="{{item}}"

with\_file:

- ~/.ssh/id\_rsa.pub

Далее приведено детальное описание операций, выполняемых этим плейбуком:

* Создает пользователя без привилегий root с именем ubuntu.
* Настраивает файл sudoers, чтобы пользователь ubuntu мог запускать команды sudo без ввода пароля.
* Добавляет на локальный компьютер открытый ключ (обычно ~/.ssh/id\_rsa.pub) в список авторизованных ключей удаленного пользователя ubuntu. Это позволит вам подключаться к каждому серверу через SSH под именем пользователя ubuntu.

Сохраните и закройте файл после добавления текста.

Затем запустите плейбук на локальном компьютере:

ansible-playbook -i hosts ~/kube-cluster/initial.yml

Выполнение команды займет от двух до пяти минут. После завершения вы увидите примерно следующий результат:

Output

PLAY [all] \*\*\*\*

TASK [Gathering Facts] \*\*\*\*

ok: [master]

ok: [worker1]

ok: [worker2]

TASK [create the 'ubuntu' user] \*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

TASK [allow 'ubuntu' user to have passwordless sudo] \*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

TASK [set up authorized keys for the ubuntu user] \*\*\*\*

changed: [worker1] => (item=ssh-rsa AAAAB3...)

changed: [worker2] => (item=ssh-rsa AAAAB3...)

changed: [master] => (item=ssh-rsa AAAAB3...)

PLAY RECAP \*\*\*\*

master : ok=5 changed=4 unreachable=0 failed=0

worker1 : ok=5 changed=4 unreachable=0 failed=0

worker2 : ok=5 changed=4 unreachable=0 failed=0

Теперь предварительная настройка завершена, и вы можете перейти к установке зависимостей Kubernetes.

Шаг 3 — Установка зависимостей Kubernetetes

В этом разделе вы научитесь устанавливать требующиеся Kubernetes пакеты уровня операционной системы с помощью диспетчера пакетов Ubuntu. Вот эти пакеты:

* Docker — среда исполнения контейнеров. Это компонент, который запускает ваши контейнеры. В настоящее время для Kubernetes активно разрабатывается поддержка других сред исполнения, в том числе [rkt](https://coreos.com/rkt/).
* kubeadm — инструмент командной строки, который устанавливает и настраивает различные компоненты кластера стандартным образом.
* kubelet — системная служба / программа, которая работает на всех узлах и обрабатывает операции на уровне узлов.
* kubectl — инструмент командной строки, используемый для отправки команд на кластер через сервер API.

Создайте в рабочем пространстве файл с именем ~/kube-cluster/kube-dependencies.yml:

nano ~/kube-cluster/kube-dependencies.yml

Добавьте в файл следующие сценарии, чтобы установить данные пакеты на ваши серверы:

- hosts: all

become: yes

tasks:

- name: install Docker

apt:

name: docker.io

state: present

update\_cache: **true**

- name: install APT Transport HTTPS

apt:

name: apt-transport-https

state: present

- name: add Kubernetes apt-key

apt\_key:

url: https://packages.cloud.google.com/apt/doc/apt-key.gpg

state: present

- name: add Kubernetes' APT repository

apt\_repository:

repo: deb http://apt.kubernetes.io/ kubernetes-xenial main

state: present

filename: 'kubernetes'

- name: install kubelet

apt:

name: kubelet=1.14.0-00

state: present

update\_cache: **true**

- name: install kubeadm

apt:

name: kubeadm=1.14.0-00

state: present

- hosts: master

become: yes

tasks:

- name: install kubectl

apt:

name: kubectl=1.14.0-00

state: present

force: yes

Первый сценарий в плейбуке выполняет следующие операции:

* Устанавливает среду исполнения контейнеров Docker.
* Устанавливает apt-transport-https, позволяя добавлять внешние источники HTTPS в список источников APT.
* Добавляет ключ apt-key репозитория Kubernetes APT для проверки ключей.
* Добавляет репозиторий Kubernetes APT в список источников APT ваших удаленных серверов.
* Устанавливает kubelet и kubeadm.

Второй сценарий состоит из одной задачи, которая устанавливает kubectl на главном узле.

Сохраните файл и закройте его после завершения.

Затем запустите плейбук на локальном компьютере:

ansible-playbook -i hosts ~/kube-cluster/kube-dependencies.yml

После завершения вы увидите примерно следующий результат:

Output

PLAY [all] \*\*\*\*

TASK [Gathering Facts] \*\*\*\*

ok: [worker1]

ok: [worker2]

ok: [master]

TASK [install Docker] \*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

TASK [install APT Transport HTTPS] \*\*\*\*\*

ok: [master]

ok: [worker1]

changed: [worker2]

TASK [add Kubernetes apt-key] \*\*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

TASK [add Kubernetes' APT repository] \*\*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

TASK [install kubelet] \*\*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

TASK [install kubeadm] \*\*\*\*\*

changed: [master]

changed: [worker1]

changed: [worker2]

PLAY [master] \*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*

ok: [master]

TASK [install kubectl] \*\*\*\*\*\*

ok: [master]

PLAY RECAP \*\*\*\*

master : ok=9 changed=5 unreachable=0 failed=0

worker1 : ok=7 changed=5 unreachable=0 failed=0

worker2 : ok=7 changed=5 unreachable=0 failed=0

После выполнения на всех удаленных серверах будут установлены Docker, kubeadm и kubelet. kubectl не является обязательным компонентом и требуется только для выполнения команд кластера. В этом контексте имеет смысл производить установку только на главный узел, поскольку вы будете запускать команды kubectl только на главном узле. Однако следует отметить, что команды kubectl можно запускать с любых рабочих узлов и на любом компьютере, где их можно установить и настроить для указания на кластер.

Теперь все системные зависимости установлены. Далее мы настроим главный узел и проведем инициализацию кластера.

Шаг 4 — Настройка главного узла

На этом шаге вы настроите главный узел. Прежде чем создавать любые плейбуки, следует познакомиться с концепциями подов и плагинов сети подов, которые будут использоваться в вашем кластере.

Под — это атомарная единица, запускающая один или несколько контейнеров. Эти контейнеры используют общие ресурсы, такие как файловые тома и сетевые интерфейсы. Под — это базовая единица планирования в Kubernetes: все контейнеры в поде гарантированно запускаются на том же узле, который назначен для этого пода.

Каждый под имеет собственный IP-адрес, и под на одном узле должен иметь доступ к поду на другом узле через IP-адрес пода. Контейнеры в одном узле могут легко взаимодействовать друг с другом через локальный интерфейс. Однако связь между подами более сложная, и для нее требуется отдельный сетевой компонент, обеспечивающий прозрачную маршрутизацию трафика между подами на разных узлах.

Эту функцию обеспечивают плагины сети подов. Для этого кластера мы используем стабильный и производительный плагин [Flannel](https://github.com/coreos/flannel).

Создайте на локальном компьютере плейбук Ansible с именем master.yml:

nano ~/kube-cluster/master.yml

Добавьте в файл следующий сценарий для инициализации кластера и установки Flannel:

- hosts: master

become: yes

tasks:

- name: initialize the cluster

shell: kubeadm init --pod-network-cidr=10.244.0.0/16 >> cluster\_initialized.txt

args:

chdir: $HOME

creates: cluster\_initialized.txt

- name: create .kube directory

become: yes

become\_user: ubuntu

file:

path: $HOME/.kube

state: directory

mode: 0755

- name: copy admin.conf to user's kube config

copy:

src: /etc/kubernetes/admin.conf

dest: /home/ubuntu/.kube/config

remote\_src: yes

owner: ubuntu

- name: install Pod network

become: yes

become\_user: ubuntu

shell: kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c114638878db11b/Documentation/kube-flannel.yml >> pod\_network\_setup.txt

args:

chdir: $HOME

creates: pod\_network\_setup.txt

Далее приведена детализация этого сценария:

* Первая задача инициализирует кластер посредством запуска kubeadm init. При передаче аргумента --pod-network-cidr=10.244.0.0/16 задается частная подсеть, из которой назначаются IP-адреса подов. Flannel использует вышеуказанную подсеть по умолчанию, и мы предпишем kubeadm использовать ту же подсеть.
* Вторая задача создает каталог .kube по адресу /home/ubuntu. В это каталоге будут храниться данные конфигурации, в том числе файлы ключа администратора, необходимые для подключения к кластеру, и адрес API кластера.
* Третья задача копирует файл /etc/kubernetes/admin.conf, сгенерированный kubeadm init в домашнем каталоге пользователя без привилегий root. Это позволит вам использовать kubectl для доступа к новому кластеру.
* Последняя задача запускает kubectl apply для установки Flannel. Синтаксис kubectl apply -f descriptor.[yml|json] предписывает kubectl создать объекты, описанные в файле descriptor.[yml|json]. Файл kube-flannel.yml содержит описания объектов, требуемых для настроки Flannel в кластере.

Сохраните файл и закройте его после завершения.

Запустите плейбук на локальной системе с помощью команды:

ansible-playbook -i hosts ~/kube-cluster/master.yml

После завершения вы увидите примерно следующий результат:

Output

PLAY [master] \*\*\*\*

TASK [Gathering Facts] \*\*\*\*

ok: [master]

TASK [initialize the cluster] \*\*\*\*

changed: [master]

TASK [create .kube directory] \*\*\*\*

changed: [master]

TASK [copy admin.conf to user's kube config] \*\*\*\*\*

changed: [master]

TASK [install Pod network] \*\*\*\*\*

changed: [master]

PLAY RECAP \*\*\*\*

master : ok=5 changed=4 unreachable=0 failed=0

Чтобы проверить статус главного узла, подключитесь к нему через SSH с помощью следующей команды:

ssh ubuntu@master\_ip

Запустите на главном узле следующую команду:

kubectl get nodes

Результат будет выглядеть следующим образом:

Output

NAME STATUS ROLES AGE VERSION

master Ready master 1d v1.24.0

В результатах показано, что главный узел master завершил выполнение всех задач инициализации и находится в состоянии готовности Ready, из которого он может принимать подключения от рабочих узлов и выполнять задачи, отправленные на сервер API. Теперь вы можете добавить рабочие узлы с локального компьютера.

Шаг 5 — Настройка рабочих узлов

Для добавления рабочих узлов в кластер нужно запустить на каждом из них отдельную команду. Эта команда предоставляет всю необходимую информацию о кластере, включая IP-адрес, порт сервера API главного узла и защищенный токен. К кластеру могут подключаться только те узлы, которые проходят проверку с защищенным токеном.

Вернитесь в рабочее пространство и создайте плейбук с именем workers.yml:

nano ~/kube-cluster/workers.yml

Добавьте в файл следующий текст для добавления рабочих узлов в кластер:

- hosts: master

become: yes

gather\_facts: **false**

tasks:

- name: get join command

shell: kubeadm token create --print-join-command

register: join\_command\_raw

- name: set join command

set\_fact:

join\_command: "{{ join\_command\_raw.stdout\_lines[0] }}"

- hosts: workers

become: yes

tasks:

- name: join cluster

shell: "{{ hostvars['master'].join\_command }} >> node\_joined.txt"

args:

chdir: $HOME

creates: node\_joined.txt

Вот что делает этот плейбук:

* Первый сценарий получает команду join, которую нужно запустить на рабочих узлах. Эта команда имеет следующий форматt: kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256:<hash>. После получения фактической команды с правильными значениями **token** и **hash** задача задает их как фактические, чтобы следующий сценарий имел доступ к этой информации.
* Второй сценарий содержит одну задачу, которая запускает команду join на всех рабочих узлах. После завершения этой задачи два рабочих узла становятся частью кластера.

Сохраните файл и закройте его после завершения.

Запустите плейбук на локальном компьютере:

ansible-playbook -i hosts ~/kube-cluster/workers.yml

После завершения вы увидите примерно следующий результат:

Output

PLAY [master] \*\*\*\*

TASK [get join command] \*\*\*\*

changed: [master]

TASK [set join command] \*\*\*\*\*

ok: [master]

PLAY [workers] \*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*

ok: [worker1]

ok: [worker2]

TASK [join cluster] \*\*\*\*\*

changed: [worker1]

changed: [worker2]

PLAY RECAP \*\*\*\*\*

master : ok=2 changed=1 unreachable=0 failed=0

worker1 : ok=2 changed=1 unreachable=0 failed=0

worker2 : ok=2 changed=1 unreachable=0 failed=0

Теперь рабочие узлы добавлены, ваш кластер полностью настроен и готов к работе, а рабочие узлы готовы к выполнению рабочих задач. Перед назначением приложений следует убедиться, что кластер работает надлежащим образом.

Шаг 6 — Проверка кластера

Иногда при установке и настройке кластера может произойти ошибка, если один из узлов отключен или имеются проблемы сетевого соединения между главным узлом и рабочими узлами. Сейчас мы проверим кластер и убедимся, что все узлы работают правильно.

Вам нужно будет проверить текущее состояние кластера с главного узла, чтобы убедиться в готовности всех узлов. Если вы отключились от главного узла, вы можете снова подключиться к нему через SSH с помощью следующей команды:

ssh ubuntu@master\_ip

Затем выполните следующую команду, чтобы получить данные о статусе кластера:

kubectl get nodes

Вы увидите примерно следующий результат:

Output

NAME STATUS ROLES AGE VERSION

master Ready master 1d v1.24.0

worker1 Ready <none> 1d v1.24.0

worker2 Ready <none> 1d v1.24.0

Если на всех ваших узлах отображается значение Ready для параметра STATUS, это означает, что они являются частью кластера и готовы к выполнению рабочих задач.

Если же на некоторых узлах отображается значение NotReady для параметра STATUS, это может означать, что настройка рабочих узлов еще не завершена. Подождите от пяти до десяти минут, а затем снова запустите команду kubectl get nodes и проверьте полученные результаты. Если для некоторых узлов еще отображается статус NotReady, вам нужно проверить и заново запустить команды, которые выполнялись на предыдущих шагах.

Теперь кластер успешно проверен, и мы запланируем запуск на кластере образца приложения Nginx.

Шаг 7 — Запуск приложения на кластере

Теперь вы можете развернуть на кластере любое приложение в контейнере. Для удобства мы развернем Nginx с помощью Deployments и Services и посмотрим, как можно развернуть это приложение на кластере. Вы можете использовать приведенные ниже команды для других приложений в контейнерах, если вы измените имя образа Docker и дополнительные параметры (такие как ports и volumes).

Запустите на главном узле следующую команду для создания развертывания с именем nginx:

kubectl create deployment nginx --image=nginx

Развертывание — это тип объекта Kubernetes, обеспечивающий постоянную работу определенного количества подов на основе заданного шаблона, даже в случае неисправности пода в течение срока службы кластера. Вышеуказанное развертывание создаст под с одним контейнером из [образа Nginx Docker](https://hub.docker.com/_/nginx/) в реестре Docker.

Запустите следующую команду, чтобы создать службу nginx, которая сделает приложение общедоступным. Для этого используется схема NodePort, которая делает под доступным на произвольном порту, который открывается на каждом узле кластера:

kubectl expose deploy nginx --port 80 --target-port 80 --type NodePort

Службы — это еще один тип объектов Kubernetes, который открывает внутренние службы кластера для внутренних и внешних клиентов. Они поддерживают запросы балансировки нагрузки на разные поды и являются неотъемлемым компонентом Kubernetes, который часто взаимодействует с другими компонентами.

Запустите следующую команду:

kubectl get services

Будет выведен текст следующего вида:

Output

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 1d

nginx NodePort 10.109.228.209 <none> 80:nginx\_port/TCP 40m

В третьей сроке результатов указан номер порта, на котором запущен Nginx. Kubernetes автоматически назначает случайный порт с номером выше 30000 и при этом проверяет, не занят ли этот порт другой службой.

Чтобы убедиться в работе всех элементов, откройте адрес http://worker\_1\_ip:nginx\_port или http://worker\_2\_ip:nginx\_port в браузере на локальном компьютере. Вы увидите знакомую начальную страницу Nginx.

Если вы захотите удалить приложение Nginx, предварительно удалите службу nginx с главного узла:

kubectl delete service nginx

Запустите следующую команду, чтобы проверить удаление службы:

kubectl get services

Результат будет выглядеть следующим образом:

Output

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 1d

Затем удалите развертывание:

kubectl delete deployment nginx

Запустите следующую команду для проверки успешности выполнения:

kubectl get deployments

Output

No resources found.

Заключение

В этом обучающем модуле вы научились успешно настраивать кластер Kubernetes в Ubuntu 18.04, используя для автоматизации Kubeadm и Ansible.

Если вы не знаете, что дальше делать с настроенным кластером, попробуйте развернуть на нем собственные приложения и службы. Далее приведен список ссылок с дополнительной информацией, которая будет вам полезна:

* [Докеризация приложений](https://docs.docker.com/engine/examples/) — детальные примеры контейнеризации приложений с помощью Docker.
* [Обзор подов](https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/) — детальное описание принципов работы подов и их отношений с другими объектами Kubernetes. Поды используются в Kubernetes повсеместно, так что понимание этой концепции упростит вашу работу.
* [Обзор развертывания](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) — обзор концепции развертывания. С его помощью проще понять принципы работы таких контроллеров как развертывания, поскольку они часто используются для масштабирования в приложениях без сохранения состояния и для автоматического восстановления поврежденных приложений.
* [Обзор служб](https://kubernetes.io/docs/concepts/services-networking/service/) — рассказывает о службах, еще одном часто используемом объекте кластеров Kubernetes. Понимание типов служб и доступных для них опций важно для использования приложений с сохранением состояния и без сохранения состояния.

Другие важные концепции, полезные при развертывании рабочих приложений: [тома](https://kubernetes.io/docs/concepts/storage/volumes/), [входы](https://kubernetes.io/docs/concepts/services-networking/ingress/) и [секреты](https://kubernetes.io/docs/concepts/configuration/secret/).

В Kubernetes имеется множество функций и возможностей. [Официальная документация Kubernetes](https://kubernetes.io/docs/) — лучший источник информации о концепциях, руководств по конкретным задачам и ссылок API для различных объектов.