Docker, Kubernetes, Minikube, etc

Основной репозиторий проекта

Установка тестового окружения (Ubuntu 16.04 Xenial)

Перед установкой тестового окружения необходимо проверить, включена ли аппаратная виртуализация (VT-x или AMD-v). Для проверки используйте простой скрипт:

```
if [ -z "$(egrep 'vmx|svm' /proc/cpuinfo)" ]; then
  echo "Virtualisation is not enabled"
else
  echo "Virtualization is enabled"
fi
```

Так же необходимо скачать и установить VirtualBox. Качаем загрузочный файл (другие версии доступны по ссылке):

```
curl -o virtualbox.deb
https://download.virtualbox.org/virtualbox/5.1.32/virtualbox-5.1_5.1.32-120294~Ubun
tu~xenial_amd64.deb
```

Устанавливаем VirtualBox:

```
sudo dpkg -i virtualbox.deb
```

Установка Docker

Удаляем предыдущие версии:

```
sudo apt-get remove docker docker-engine docker.io
```

Скачиваем установочный пакет Docker:

```
curl -o docker.deb
https://download.docker.com/linux/ubuntu/dists/xenial/pool/stable/amd64/docker-ce_1
7.03.2~ce-0~ubuntu-xenial_amd64.deb
```

> Если вы используете другую версию Ubuntu, перейдите по ссылке, выберите вашу версию дистрибутива, перейдите в папку /pool/stable и выберите вашу архитектуру (например, amd64). Скачайте `.deb` файл Docker.

Устанавливаем Docker из загруженного .deb файла, выполнив:

```
sudo dpkg -i ./docker.deb
```

(измените путь/название файла при необходимости)

Установка kubectl

Скачиваем исполняемый файл kubectl:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd
64/kubectl
```

Выдаем права на запуск:

```
chmod +x ./kubectl
```

Перемещаем исполняемый файл в стандартную директорию для исполняемых файлов в Ubuntu:

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

Установка minikube

Скачиваем исполняемый файл minikube:

```
curl -Lo minikube
https://github.com/kubernetes/minikube/releases/download/v0.25.0/minikube-linux-amd
64
```

Выдаем права на запуск:

```
chmod +x ./minikube
```

Перемешаем исполняемый файл в стандартную директорию для исполняемых файлов в Ubuntu:

```
sudo mv ./minikube /usr/local/bin/minikube
```

Проверим установку minikube:

```
$ minikube start
Starting local Kubernetes v1.9.0 cluster...
Starting VM...
Downloading Minikube ISO
 142.22 MB / 142.22 MB [============================================] 100.00% 0s
Getting VM IP address...
Moving files into cluster...
Downloading localkube binary
 162.41 MB / 162.41 MB [============================================] 100.00% 0s
 0 B / 65 B [----------------------------------------------------------]   0.00%
 65 B / 65 B [==========================================================] 100.00%
0sSetting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
Loading cached images from config file.
```

Получим информацию о статусе minikube-кластера:

```
$ minikube status
minikube: Running
cluster: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100
```

Проверим корректность работы kubectl:

```
$ kubectl get nodes
NAME       STATUS    ROLES     AGE       VERSION
minikube   Ready     <none>    9d        v1.9.0
```

Для вывода информации и управления кластером используется панель управления minikube (minikube dashboard). Запускается она одноименной командой:

```
minikube dashboard
```

В случае необходимости, minikube-кластер можно остановить данной командой:

```
$ minikube stop
Stopping local Kubernetes cluster...
Machine stopped.
```

Так же создайте аккаунт в Docker Hub.

Docker

Docker - это откртая платформа для разработки, доставки и эксплуатации приложений. Docker предоставляет возможность запускать изолированные на уровне системы приложения, причем делать это намного быстрее и проще, чем с помощью виртуальных машин.

Docker состоит из трех компонентов:

1. Образы (images) - read-only шаблон, на основе которого создается контейнер. Бывают базовыми (например, образы операционных систем) и дочерние (образы, построенные на базовых и имеющие дополнительный функционал)
2. Контейнеры - процесс (приложение), изолированный в окружении, содержащим все необходимое для работы этого процесса. Контейнеры могут быть созданы (из образа), запущены, остановлены, перенесены или удалены. Каждый контейнер является безопасной переносимой платформой для приложения. Контейнеры - это основная компонента работы Docker.
3. Реестр (register) - хранилище образов. Бывают публичными и приватными. Позволяют скачивать и загружать образы. Самым большим публичным реестром является Docker Hub

Docker использует клиент-серверную архитектуру. Клиент предоставляет интерфейс, получает команды от пользователя и передает их демону Docker, который создает, запускает и распределяет контейнеры. Клиент и сервер могут работать в одной системе, либо клиент может подключаться к удаленному серверу, используя RESTfull API или сокеты.

Docker-контейнеры используют так называемую "объединенную файловую систему" (union file sistem), состоящую из нескольких слоев. При этом одинаковые слои не дублируются в памяти хост-машины, а используются совместно (разумеется, с выполнением условия изоляции), что позволяет существенно сократить объемы памяти, необходимые для хранения контейнеров.

Docker следует принципу "один контейнер - один процесс". Это значит, что каждый контейнер должен содержать только один процесс(приложение), взаимодействующий с другими приложениями по сети. Помимо этого, принято считать, что docker-контейнер - это инструмент, и должен использоваться только по необходимости. Это значит, что в docker-контейнере не должны храниться какие-либо данные, являющиеся результатом работы контейнера.

Основы Docker

Запустим простейший docker-контейнер:

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

С помощью команды `docker run` мы запустили на выполнение контейнер hello-world, единственным предназначением которого было выведение на экран текста приветствия. По

завершению работы процесса контейнер автоматически прекратил свою работу.

Попробуем кое-что более интересное. Для этого скачаем из официального реестра Docker Hub образ BusyBox - набора утилит для командной строки Linux.

Загрузим образ BusyBox:

```
docker pull busybox
```

Получим список загруженных образов:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
busybox             latest              f6e427c148a7        13 days ago
1.15 MB
hello-world         latest              f2a91732366c        3 months ago
1.85 kB
```

Запустим скрипт на выполнение в контейнере BusyBox:

```
$ docker run busybox echo "hello from busybox"
hello from busybox
```

Для вывода информации о запущенных процессов используется команда

```
docker ps
```

Для вывода информации о запущенных ранее процессах добавьте ключ `-a` Так же контейнер можно запускать в интерактивном режиме, используя флаг `-it`. Запустим на выполнение bash-оболочку в контейнере BusyBox в интерактивном режиме:

```
$ docker run -it busybox sh
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ # whoami
root
```

Для удаления контейнера воспользуемся командой `docker rm` с указанием идентификатора контейнера, например

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                      PORTS               NAMES
```

```
ecfe781b05ec        busybox             "sh"                        2 minutes ago
Exited (0) About a minute ago                       cocky_nobel
5fac160babc1        busybox             "echo 'hello from ..."   11 minutes ago
Exited (0) 11 minutes ago                           dreamy_bardeen
$ docker rm ecfe781b05ec 5fac160babc1
ecfe781b05ec
5fac160babc1
```

Так же можно удалить все контейнеры, воспользовавшись командой `docker rm $(docker ps -a -q -f status=exited)`

Удаление ненужных образов выполняется с помощью команды `rmi`:

```
$ docker rmi busybox
Untagged: busybox:latest
Untagged:
busybox@sha256:2107a35b58593c58ec5f4e8f2c4a70d195321078aebfadfbfb223a2ff4a4ed21
Deleted: sha256:f6e427c148a766d2d6c117d67359a0aa7d133b5bc05830a7ff6e8b64ff6b1d1d
Deleted: sha256:c5183829c43c4698634093dc38f9bee26d1b931dedeba71dbee984f42fe1270d
```

Создание контейнера

Для создание контейнера зарегистрируйтесь на Docker Hub. После этого скачайте репозиторий с тестовым приложением и перейдите в директорию `flask_base_app`.

Основной файл для создания репозитория называется Dockerfile. Его содержимое выглядит следующим образом:

```
FROM python:3-onbuild

# Порт, который необходимо открыть для внешнего доступа
EXPOSE 5000

# Запускаем приложение на выполнение
CMD ["python", "./app.py"]
```

Секция `FROM` указывает на родительский контейнер. В данном случае это контейнер, содержащий оптимизированный для запуска python-приложений в продакшне. Секция `EXPOSE` устанавливает порт, открытый для внешнего взаимодействия. Секция `CMD` указывает, какие команды необходимо запустить при старте контейнера.

Помимо приложения и файла для сборки контейнера, в папке находится файл `requirements.txt`, который содержит список python-библиотек, необходимых для запуска приложения.

> Название контейнера `antonkravtsevich/base_flask_app` состоит из ID пользователя на Dockerhub (antonkravtsevich) и названия самого контейнера

(base_flask_app). Здесь и далее замените ID пользователя своим, чтобы иметь возможность загрузить контейнер на Dockerhub.

Соберем контейнер:

```
$ docker build -t antonkravtsevich/base_flask_app .
Sending build context to Docker daemon 4.096 kB
Step 1/3 : FROM python:3-onbuild
# Executing 3 build triggers...
Step 1/1 : COPY requirements.txt /usr/src/app/
Step 1/1 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Running in 75ec8b4fa72a
Collecting Flask==0.12.2 (from -r requirements.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting itsdangerous>=0.21 (from Flask==0.12.2->-r requirements.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.4 (from Flask==0.12.2->-r requirements.txt (line 1))
  Downloading Jinja2-2.10-py2.py3-none-any.whl (126kB)
Collecting click>=2.0 (from Flask==0.12.2->-r requirements.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting Werkzeug>=0.7 (from Flask==0.12.2->-r requirements.txt (line 1))
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->Flask==0.12.2->-r requirements.txt
(line 1))
  Downloading MarkupSafe-1.0.tar.gz
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, click, Werkzeug,
Flask
  Running setup.py install for itsdangerous: started
    Running setup.py install for itsdangerous: finished with status 'done'
  Running setup.py install for MarkupSafe: started
    Running setup.py install for MarkupSafe: finished with status 'done'
Successfully installed Flask-0.12.2 Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1
click-6.7 itsdangerous-0.24
Step 1/1 : COPY . /usr/src/app
 ---> 60f8cda970ae
Removing intermediate container ac905ca52934
Removing intermediate container 75ec8b4fa72a
Removing intermediate container 82032170dca7
Step 2/3 : EXPOSE 5000
 ---> Running in 6a3a1ea19281
 ---> 7595e4f26987
Removing intermediate container 6a3a1ea19281
Step 3/3 : CMD python ./app.py
 ---> Running in 66680a2bbaa9
 ---> 147dcd57f85c
Removing intermediate container 66680a2bbaa9
Successfully built 147dcd57f85c

$ docker images
REPOSITORY                        TAG                 IMAGE ID            CREATED
SIZE
antonkravtsevich/base_flask_app   latest              147dcd57f85c        59
seconds ago      697 MB
<none>                            <none>              becb73ed6142        2 minutes
ago       688 MB
```

```
python                              3-onbuild           badd7f8f9d5a        25 hours
ago         688 MB
hello-world                         latest              f2a91732366c        3 months
ago         1.85 kB
```

Образ `antonkravtsevich/base_flask_app` появился в списке доступных. Запустим этот контейнер:

```
$ docker run -p 8888:5000 antonkravtsevich/base_flask_app
      * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Флаг -p позволяет указывать проброс портов (в данном случае - с 8888 на 5000). С помощью этой команды можно открыть порты контейнера для доступа извне. Перейдите в браузере по адресу http://127.0.0.1:8888. Если контейнер был запущен успешно, вы получите страницу с текстом `Hello from container!`.

Загрузим свой образ на Docker Hub. Для этого нам сперва нужно авторизоваться, введя логин и пароль, указанные при регистрации на Docker Hub, в консоли:

```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you dont have
a Docker ID, head over to https://hub.docker.com to create one.
Username: antonkravtsevich
Password:
Login Succeeded
```

Затем отправим собранный контейнер в Docker Hub:

```
$ docker push antonkravtsevich/base_flask_app
The push refers to a repository [docker.io/antonkravtsevich/base_flask_app]
278d585f6977: Pushed
7cd0e760a754: Pushed
2bf8865989f1: Pushed
90d4e4e9eebd: Mounted from library/python
aec4f1507d85: Mounted from library/python
a4a7a3673769: Mounted from library/python
325a22db58ea: Mounted from library/python
6e1b48dc2ccc: Mounted from library/python
ff57bdb79ac8: Mounted from library/python
6e5e20cbf4a7: Mounted from library/python
86985c679800: Mounted from library/python
8fad67424c4e: Mounted from library/python
latest: digest:
sha256:4e738835482a4fc31a32174a1ce85eea8078e349a024d057a3f5308016fcdea7 size: 2839
```

Теперь вы сможете загрузить этот образ на любую рабочую машину, воспользовавшись командой `docker pull`.

Это все, что необходимо знать для запуска тестового кластера kubernetes.

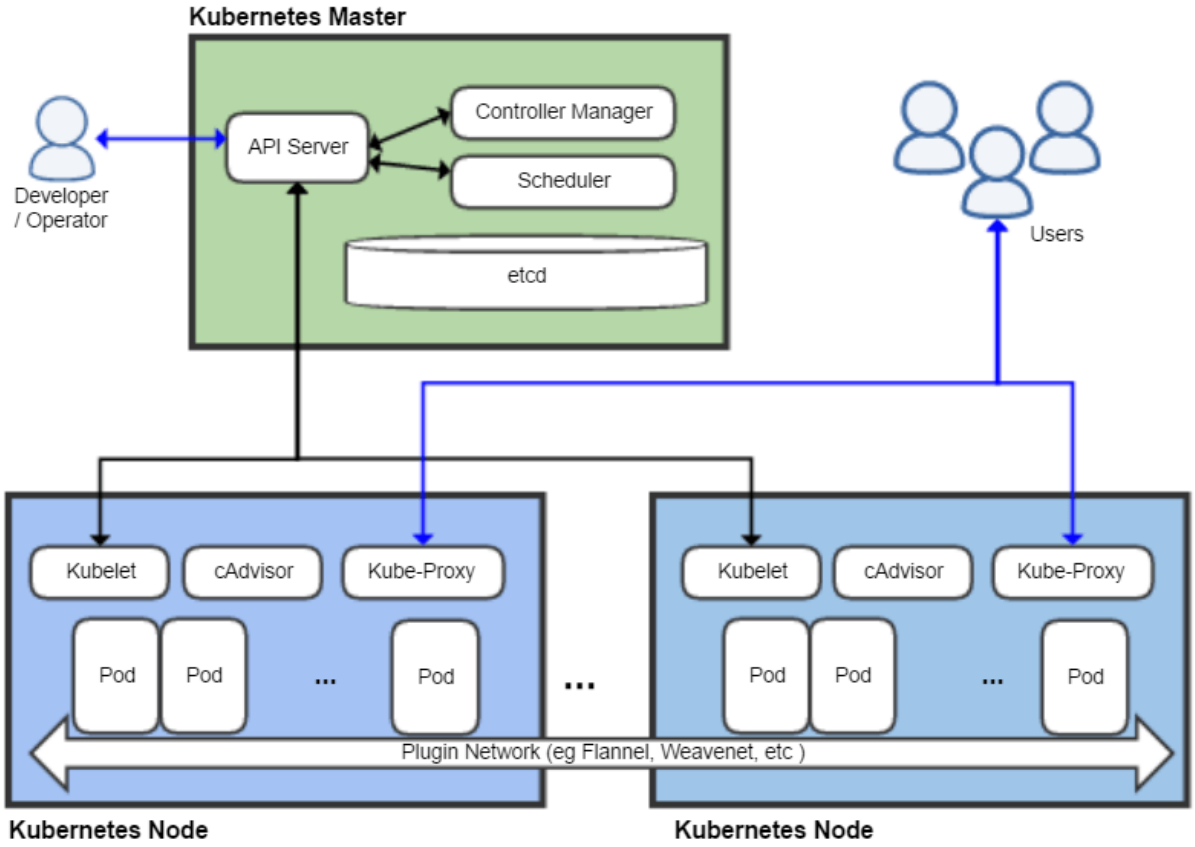
Kubernetes

Kubernetes - открытое ПО для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями (т.н. оркестровка). Kubernetes позволяет управлять кластером контейнеров Linux как единой системой. Проект был начат комманией Google, а затем передан под управление Cloud Native Computing Foundation.

Основные концепции кластера Kubernetes:

1. **Ноды** (*nodes*) - реальные физические или виртуальные машины, являющиеся участниками kubernetes-кластера. Каждый нод содержит сервисы для запуска приложений (Docker, например), а так же компоненты, предназначенные для централизованного управления нодой.
2. **Поды** (*pods*) - группа контейнеров с общими разделами (например, приложение+БД), запускаемая как единое целое. Являются основной рабочей еденицей kubernetes-кластера. Каждому поду гарантируется запуск на одной физической машине и выделение уникального в пределах кластера виртуального IP-адреса, что позволяет использовать предопределенные номера портов без опасности конфликта.
3. **Сервисы** (*services*) - совокупность подов и политики доступа к нему. Предоставляют внешнюю выходную точку для общения с приложениями, запущенными на подах. Обычно являются общими для определенной категории подов. Типичный пример - несколько подов с запущенным backend-приложением имеют один и тот же сервис с публичным адресом, и все запросы к этому сервису распределяются между подами с использованием балансировки нагрузки.
4. **Разделы** (*volumes*) - каталоги или сетевые диски (могут содержать данные), которые доступны контейнерам пода.
5. **Deployment** - объединение подов и Replica Controller, отслеживающего поды и запускающего/удаляющего их при необходимости. Отвечает за то, чтобы в любой момент времени было запущенно указанное количество подов данного типа.

Структура кластера Kubernetes



В kubernetes-кластере существуют главные (master) и рабочие (worker) ноды. Наиболее стабильными являются кластеры с несколькими главными и несколькими рабочими нодами.

Компоненты главной ноды:

1. **API сервер** принимает управляющие команды из консоли или веб-интерфейса.
2. **Controller Manager** - менеджер контроллеров, отслеживает запуск и работу других контроллеров. Включает:
   1. **Node Controller** следит за состоянием нод
   2. **Replication Controller** отвечает за запуск нужного количество подов в любой момент времени
   3. **Endpoint Controller** отвечает за связывание сервисов и подов
   4. **Service Account & Token Controllers** создает аккаунты по умолчанию/выдает токены для доступа новых пользователей
3. **Sheduler** отслеживает создаваемые поды и выбирает ноду, на которой они будут запущены.
4. **etcd** - высокодоступное, надежное key-value хранилище, хранит резервные данные кластера.

Компоненты рабочей ноды:

1. **kubelet** - принимает команды от главных узлов, управляет контейнерами и подами, отслеживает и обеспечивает их работоспособность. Рабочий агент.
2. **kube-proxy** - ПО, эмулирующее виртуальную, независимую от физической сеть между подами в кластере.
3. **Container Runtime** - контейнеры, развернутые на ноде.

В качестве основного интерфейса работы с кластером (помимо веб-интерфейса) используется утилита командной строки **kubectl**.

Запуск автономного приложения в kubernetes-кластере

Для тестирования мы будем использовать локальный кластер, развернутый на рабочей машине с помощью minikube. Minikube позволяет с помощью одной команды запустить полноценный кластер, состоящий из одной ноды, и настроить kubectl для работы с ним.

Запустим minikube и проверим работоспособность кластера:

```
$ minikube start
Starting local Kubernetes v1.9.0 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
Loading cached images from config file.

$ minikube status
minikube: Running
cluster: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100
```

Откроем панель управления minikube, воспользовавшись командой

```
$ minikube dashboard
Opening kubernetes dashboard in default browser...
В текущем сеансе браузера создано новое окно.
```

В браузере откроется веб-интерфейс для работы с кластером, с помощью которого можно отслеживать состояние элементов кластера, а так же создавать новые объекты. Однако намного проще использовать утилиту kubectl.

Для описания объектов kubectl используются файлы в формате `.yaml`. Создадим такой файл для нашего приложения:

bfa.yaml

```
# Версия API, по которой работает API сервер
apiVersion: extensions/v1beta1
# Тип объекта kubernetes
kind: Deployment
# Метаданные, описывающие данный объект
metadata:
```

```
  # название объекта
  name: bfa
# Данные, на основании которых будет создан объект
spec:
  # количество реплик подов
  replicas: 3
  # описание пода
  template:
    # метаданные шаблона
    metadata:
      # labels позволяют создавать пары ключ-значение, по которым
      # в дальнейшем можно проводить выборки объектов
      labels:
        # будет создана пара app: flask_app
        app: flask_app
    # данные, на основании которого будет создан под
    spec:
      # информация о контейнере
      containers:
        # название контейнера
      - name: bfa
        # образ, на основе которого будет создан контейнер
        image: antonkravtsevich/base_flask_app
        # порты, которые необходимо открыть
        ports:
          # основной порт контейнера
        - containerPort: 5000
```

Создадим deployment на основании этого `.yaml` файла:

```
$ kubectl create -f bfa.yaml
deployment "bfa" created
```

Kubectl сообщит о том, что deployment создан, однако это еще не означает, что приложение заработало. Информацию о всех подах кластера можно получить, выполнив команду

```
$ kubectl get pods
NAME                   READY     STATUS              RESTARTS   AGE
bfa-7f76f8b449-bg7zq   0/1       ContainerCreating   0          38s
bfa-7f76f8b449-d7zcx   0/1       ContainerCreating   0          38s
bfa-7f76f8b449-hrqtr   0/1       ContainerCreating   0          38s
```

ContainerCreating в статусе контейнера означает, что в данный момент поды находятся на этапе создания контейнера. Более подробную информацию о состоянии контейнера можно получить, воспользовавшись командой

```
$ kubectl describe pod bfa-7f76f8b449-bg7zq
Name:           bfa-7f76f8b449-bg7zq
Namespace:      default
Node:           minikube/192.168.99.100
```

```
Start Time:     Fri, 16 Mar 2018 02:00:33 +0300
Labels:         app=flask_app
                pod-template-hash=3932946005
Annotations:    <none>
Status:         Pending
IP:
Controlled By:  ReplicaSet/bfa-7f76f8b449
Containers:
  bfa:
    Container ID:
    Image:          antonkravtsevich/base_flask_app
    Image ID:
    Port:           5000/TCP
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-4c7qg (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          False
  PodScheduled   True
Volumes:
  default-token-4c7qg:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-4c7qg
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     <none>
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              2m    default-scheduler  Successfully assigned
bfa-7f76f8b449-bg7zq to minikube
  Normal  SuccessfulMountVolume  2m    kubelet, minikube  MountVolume.SetUp
succeeded for volume "default-token-4c7qg"
  Normal  Pulling                2m    kubelet, minikube  pulling image
"antonkravtsevich/base_flask_app"
```

bfa-7f76f8b449-bg7zq - это ID пода, полученный в результате команды `kubectl get pods`.

События, происходящие/произошедшие в контейнере, записаны в самом низу вывода команды describe. Как видно, в данный момент под занят загрузкой образа `antonkravtsevich/base_flask_app`. Эта операция займет около пяти минут при стабильном интернет-соединении.

После того, как загрузка будет завершена, вывод команды `get pods` примет следующий вид:

```
$ kubectl get pods
NAME                   READY     STATUS    RESTARTS   AGE
```

```
bfa-7f76f8b449-bg7zq   1/1       Running   0          10m
bfa-7f76f8b449-d7zcx   1/1       Running   0          10m
bfa-7f76f8b449-hrqtr   1/1       Running   0          10m
```

Теперь поды запущены и приложение работает, однако только внутри виртуальной сети кластера kubernetes. Для того, чтобы получить к нему доступ, необходимо создать сервис, который будет являться выходной точкой для этого приложения.

Создадим `.yaml` файл с описанием сервиса:

bsa-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: bfa-service
  labels:
    app: flask_app
    run: flask_app-service
spec:
  # Метод для открытия порта
  type: NodePort
  ports:
  - port: 5000
    protocol: TCP
  # Селектор, по которому выбираются целевые поды
  selector:
    app: flask_app
```

С помощью селектора производится выборка подов, для которых этот сервис должен будет служить выходной точкой.

Создадим сервис:

```
$ kubectl create -f bfa-service.yaml
service "bfa-service" created
```

Проверим его наличие в списке сервисов и получим дополнительную информацию:

```
$ kubectl get services
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
bfa-service   NodePort    10.96.94.20   <none>        5000:32418/TCP   1m
kubernetes    ClusterIP   10.96.0.1     <none>        443/TCP          9d

kubectl describe service bfa-service
Name:                   bfa-service
Namespace:              default
Labels:                 app=flask_app
                        run=flask_app-service
Annotations:            <none>
```

```
Selector:                 app=flask_app
Type:                     NodePort
IP:                       10.96.94.20
Port:                     <unset>  5000/TCP
TargetPort:               5000/TCP
NodePort:                 <unset>  32418/TCP
Endpoints:                172.17.0.5:5000,172.17.0.6:5000,172.17.0.8:5000
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

С помощью созданного сервиса все запросы, приходящие на порт 32418 (выделяется динамически) minikube будут автоматически перенаправлены на порт 5000 одного из подов. Попробуем получить к ним доступ. Для этого запросим IP-адрес кластера minikube:

```
$ minikube ip
192.168.99.100
```

И перейдем в браузере по адресу `192.168.99.100:32418` (порт может отличаться)

Если развертывание всех компонентов прошло правильно, вы получите страницу с сообщением `Hello from container!`.

Запуск составного приложения в kubernetes-кластере

Для этого примера будет использовано стандартное приложение, состоящее из фронт-энда и базы данных. Воспользуемся готовым приложением из данного репозитория. Данное приложение позволяет подписаться на рассылку, добавив свой e-mail в список. Все необходимые образы контейнеров уже находятся в Docker Hub, все что нам нужно - развернуть их в кластере.

Для начала запустим базу данных.

База данных создается с помощью Deployment, описанного следующим `.yaml` файлом:

rsvp-db.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: rsvp-db
spec:
  replicas: 1
  template:
    metadata:
      labels:
        appdb: rsvpdb
    spec:
      containers:
```

```
      - name: rsvpd-db
        image: mongo:3.3
        env:
          # Переменная окружения, хранящая название базы данных
        - name: MONGODB_DATABASE
          value: rsvpdata
        ports:
        - containerPort: 27017
```

Создадим под:

```
$ kubectl create -f rsvp-db.yaml
deployment "rsvp-db" created
```

Создадим сервис для того, чтобы открыть доступ к базе данных:

rsvp-db-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  labels:
    app: rsvpdb
spec:
  ports:
  - port: 27017
    protocol: TCP
  selector:
    appdb: rsvpdb
```

```
$ kubectl create -f rsvp-db-service.yaml
service "mongodb" created
```

Теперь сервис запущен. Получим информацию о deployment-ах, подах и сервисах:

```
$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bfa       3         3         3            3           2h
rsvp-db   1         1         1            1           6m

$ kubectl get pods
NAME                      READY     STATUS    RESTARTS   AGE
bfa-7f76f8b449-bg7zq      1/1       Running   0          2h
bfa-7f76f8b449-d7zcx      1/1       Running   0          2h
bfa-7f76f8b449-hrqtr      1/1       Running   0          2h
rsvp-db-759bcb695-fx2zz   1/1       Running   0          4m

$ kubectl get services
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
```

```
bfa-service   NodePort    10.96.94.20    <none>        5000:32418/TCP   1h
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP          9d
mongodb       ClusterIP   10.110.234.0   <none>        27017/TCP        1m
```

Создадим и запустим deployment приложения, отвечающего за фронт-энд:

rsvp-web.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: rsvp
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: rsvp
    spec:
      containers:
      - name: rsvp-app
        image: teamcloudyuga/rsvpapp
        env:
        # создание переменной окружения, которое будет хранить ip-адрес базы данных
        - name: MONGODB_HOST
          # в качестве значения передается название сервиса
          value: mongodb
        ports:
        - containerPort: 5000
          # имя порта
          name: web-port
```

> Тип порта `containerPort` позволяет использовать имя порта вместо жесткого нумерованного значения.

В приложении, отвечающем за фронт, для подключения к базе данных используется следующий код:

```
MONGODB_HOST=os.environ.get('MONGODB_HOST', 'localhost')
client = MongoCLient(MONGODB_HOST, 27017)
```

Этот код получает IP-адрес базы данных из переменной окружения.

Создадим deployment:

```
$ kubectl create -f rsvp-web.yaml
deployment "rsvp" created
```

```
$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bfa       3         3         3            3           2h
rsvp      1         1         1            1           18s
rsvp-db   1         1         1            1           11m
```

Создадим сервис для фронтэнда:

rsvp-web-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: rsvp
  labels:
    apps: rsvp
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: web-port
    protocol: TCP
  selector:
    app: rsvp
```

> В секции `targetPort` указано имя порта, которое мы дали рабочему порту нашего приложения. Подобная привязка позволит менять выходной порт приложения без необходимости создавать новый сервис.

```
$ kubectl create -f rsvp-web-service.yaml
service "rsvp" created

$ kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
bfa-service   NodePort    10.96.94.20     <none>        5000:32418/TCP   1h
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP          9d
mongodb       ClusterIP   10.110.234.0    <none>        27017/TCP        12m
rsvp          NodePort    10.107.161.11   <none>        80:32238/TCP     18s
```

Minikube транслирует все данные, полученные на порт 32238, на восьмидесятый порт сервиса rsvp. Попробуем получить к нему доступ:

```
$ minikube ip
192.168.99.100
```

Перейдем в браузере по адресу 192.168.99.100:32238. Откроется главная страница приложения, на которой можно добавлять новые e-mail'ы.

При увеличении нагрузки можно увеличить количество запущенных подов, используя команду `scale`:

```
$ kubectl get pods
NAME                      READY     STATUS    RESTARTS   AGE
bfa-7f76f8b449-bg7zq      1/1       Running   0          2h
bfa-7f76f8b449-d7zcx      1/1       Running   0          2h
bfa-7f76f8b449-hrqtr      1/1       Running   0          2h
rsvp-876876b6c-d6wgz      1/1       Running   0          11m
rsvp-db-759bcb695-fx2zz   1/1       Running   0          22m

$ kubectl scale --replicas=4 -f rsvp-web.yaml
deployment "rsvp" scaled

$ kubectl get pods
NAME                      READY     STATUS    RESTARTS   AGE
bfa-7f76f8b449-bg7zq      1/1       Running   0          2h
bfa-7f76f8b449-d7zcx      1/1       Running   0          2h
bfa-7f76f8b449-hrqtr      1/1       Running   0          2h
rsvp-876876b6c-5pjm9      1/1       Running   0          49s
rsvp-876876b6c-9czqc      1/1       Running   0          49s
rsvp-876876b6c-d6wgz      1/1       Running   0          13m
rsvp-876876b6c-rrgc8      1/1       Running   0          49s
rsvp-db-759bcb695-fx2zz   1/1       Running   0          24m
```