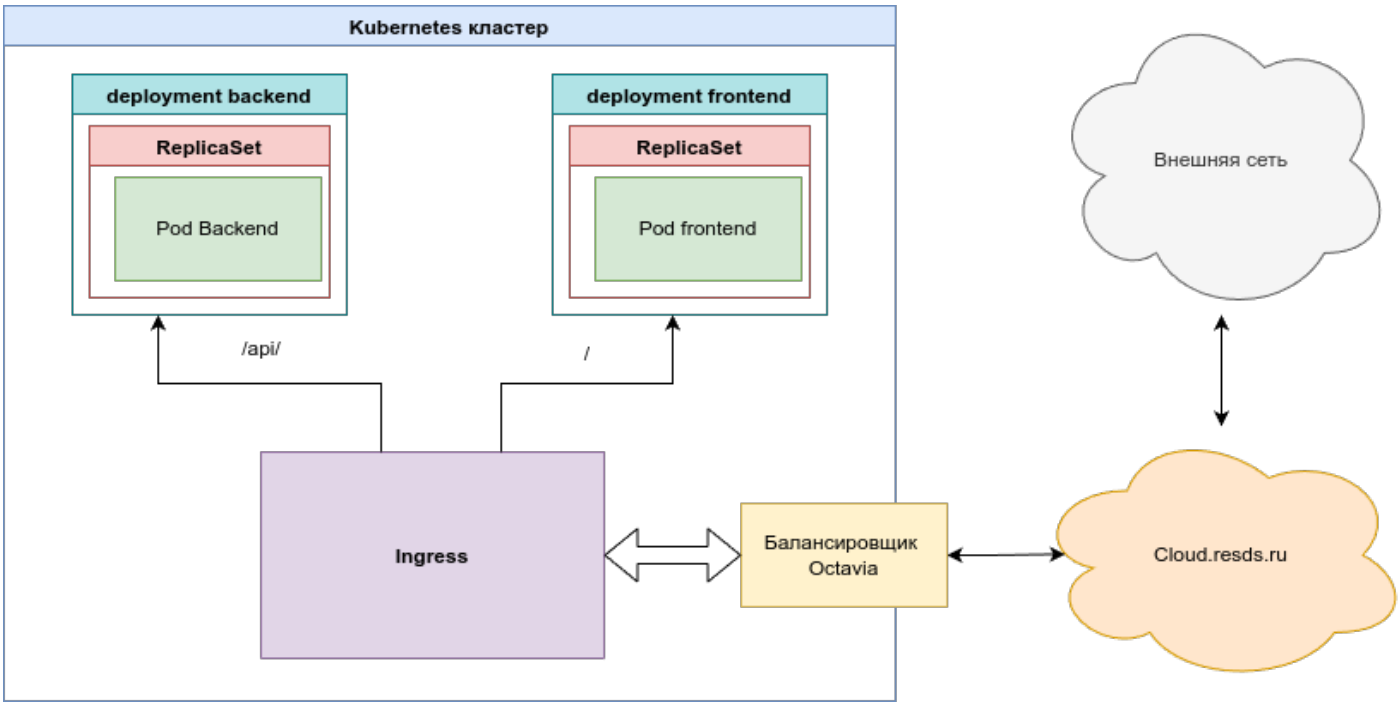
Автоматизация развертывания приложений. Kubernetes.

Схема стенда



Подготовка окружения

1. Загрузите версию `v1.25.9` с помощью команды:

   ```
   curl -LO https://dl.k8s.io/release/v1.25.9/bin/linux/amd64/kubectl
   ```

2. Сделайте бинарный файл kubectl исполняемым:

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

3. Переместите бинарный файл в директорию из переменной окружения PATH:

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

4. Убедитесь, что установлена последняя версия:

```
kubectl version --client
```

```
cloudadmin@kubectl:~$ kubectl version --client
WARNING: This version information is deprecated and will be replaced with the output from kubectl ve
n --short.  Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.9", GitCommit:"a1a87a0a2bcd605
20c6b0e618a8ab7d117d4", GitTreeState:"clean", BuildDate:"2023-04-12T12:16:51Z", GoVersion:"go1.19.8"
mpiler:"gc", Platform:"linux/amd64"}
Kustomize Version: v4.5.7
```

5. Установите CLI для OpenStack:

```
sudo pip install cryptography==3.3.2
sudo pip install python-openstackclient
sudo pip install python-cinderclient python-magnumclient python-octaviaclient
```

6. Загрузите учетные данные, для получения доступа к API. Для этого необходимо авторизоваться на `cloud.resds.ru` в правом верхнем углу нажмите на клавишу "Загрузите файл OpenStack RC" и выберете в выпадающем меню "OpenStack RC-файл" и переместите этот файл в локальную директорию пользователя с названием "openstack-client.sh"
7. Добавьте в переменные среды полученные данные из прошлого шага:

```
source ~/openstack-client.sh
```

> При добавлении переменных среды, будет запрошен пароль, и добавлен в переменные среды на время сессии

8. Для проверки работоспособности клиента можно попробовать получить список вычислительных узлов в проекте

```
openstack server list
```

Также можно увидеть список кластер k8s

```
openstack coe cluster list
```

9. Также добавим автодополнение для bash:

```
echo 'source <(kubectl completion bash)' >>~/.bashrc
source ~/.bashrc
```

Создание кластера

1. Авторизуйтесь `cloud.resds.ru`
2. Перейдите в `Проект` -> `Container Infra` -> `Clusters`
3. Нажмите `Create Cluster`
4. Задайте название кластера в поле `Cluster Name`, название кластера `Training-cluster-{Фамилия}`, как пример `Training-cluster-Tarabanov`
5. В `Cluster Template` выберите пункт `c39-kube-1.25-oct-ing`
6. Зона доступности кластера `nova`
7. Выберите ключевую пару
8. Как размер кластера выберете 1 мастер узел и 2 рабочих узла
9. Нажмите отправить
10. После отправки необходимо дождаться создания кластера, это может занять некоторое время. Кластер можно считать созданным после смены статуса на `CREATE_COMPLETE` и `Health Status` в `HEALTHY`

Работа с кластером

1. Перейдите на предварительно настроенный узел
2. Добавьте переменные среды
3. Получите конфигурационный файл для подключения к кластеру с помощью CLI

```
openstack coe cluster config {Название кластера созданного вами}
```

```
cloudadmin@kubectl:~$ openstack coe cluster config Training-cluster-Tarabar
export KUBECONFIG=/home/cloudadmin/config
```

```
export KUBECONFIG=/home/cloudadmin/config
```

4. Проверьте работу kubectl

```
kubectl get nodes
```

```
cloudadmin@kubectl:~$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSI
training-cluster-tarabanov-tnfhmviiv554-master-0   Ready    master   11m     v1.25
training-cluster-tarabanov-tnfhmviiv554-node-0     Ready    <none>   6m40s   v1.25
training-cluster-tarabanov-tnfhmviiv554-node-1     Ready    <none>   6m26s   v1.25
```

5. Создания ингреса для нашего приложения:
   Ингрес будет использовать облачный балансировщик Octavia, который реализуют балансировщик на базе облачной платформы OpenStack. Для начала необходимо посмотреть существующие балансировщики в облачной платформе с помощью команды:

```
openstack loadbalancer list
```

   Создадим файл с конфигурацией ингреса ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: openstack
    octavia.ingress.kubernetes.io/internal: "false"
  name: octavia-ingress
spec:
  rules:
  - host: test.com
    http:
      paths:
      - backend:
          service:
            name: sample-front
            port:
              number: 80
        path: /
        pathType: Prefix
```

Данный фрагмент конфигурации определяет параметры создаваемого ингресса в Kubernetes. Этот конфиг определяет следующие основные параметры:

- apiVersion: Версия API, используемая для определения структуры конфигурации.
- kind: Тип объекта, в данном случае, это ингресс.
- metadata: Метаданные объекта, такие как имя и пространство имен.
- annotations: Аннотации предоставляют дополнительные метаданные об ингрессе.
- spec: Определяет спецификацию правил ингресса, включая хосты и пути, а также их бекенды. Для данного ингресса мы определили, что будет использоваться домен test.com и для него будет использоваться бекенд сервисом `sample-front` и посылать запросы на 80 порт сервиса, переход будет осуществляться при любом запросе к домену `http://test.com/`. Чтобы применить выполненную команду, необходимо ввести команду:

```
kubectl apply -f ingress.yaml
```

После выполнения данной команды начнется создание балансировщика в платформе OpenStack, чтобы отследить создание балансировщика можно выполнять команду:

```
openstack loadbalancer list
```

Если после создания балансировщика появилась ошибка в столбце `provisioning_status` у вашего балансировщика, выполните команду:

```
openstack loadbalancer failover <loadbalancer id>
```

> Если все еще находится в состоянии ошибки и не выходит после этой команды, пишите @tarabanov

Пока балансировщик никуда не обращается, давайте создадим поды и сервисы для нашего веб приложения.
Для этого начнем создание конфигурационного файла для фронтенда приложения, для этого начнем с создания файла `sample-web-deployments.yaml`

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-front
  labels:
    app: sample-front
spec:
  selector:
    matchLabels:
      app: sample-front
  template:
    metadata:
      labels:
        app: sample-front
    spec:
      containers:
      - name: sample-front
        image: registry.gitlab.resds.ru/itt/sample-project:front
        ports:
        - containerPort: 3000
```

Этот фрагмент конфигурации представляет собой определение деплоймента в Kubernetes, который управляет подами для запуска и масштабирования приложения. Давайте разберем его по шагам:

- apiVersion: apps/v1 и kind: Deployment: Эти строки определяют, что мы создаем деплоймент в Kubernetes API версии 1 для управления подами.
- metadata: В этом разделе определяется метаданные деплоймента, такие, как его имя и метки (labels), которые могут использоваться для идентификации и поиска этого ресурса в Kubernetes кластере.
- spec: Этот раздел определяет спецификацию деплоймента, включая:
  - selector: Определяет, какие поды должны быть управляемы деплойментом. В данном случае, деплоймент будет управлять подами, которые имеют метку `app: sample-front`
  - template: Определяет шаблон для создания новых подов. Внутри template указываются метаданные и спецификация контейнера.
  - metadata.labels: Это метки, которые будут присвоены создаваемым подам. В данном случае, создаваемые поды будут иметь метку app: sample-front, что соответствует селектору деплоймента.
  - `spec.containers`: Определяет контейнеры, которые будут запущены в подах. В данном случае, определен только один контейнер:
    - `name: sample-front`: Это имя контейнера.
    - `image: registry.gitlab.resds.ru/itt/sample-project:front`: Это образ контейнера, который будет загружен и запущен в поде.
    - `ports`: Это список портов, которые контейнер будет прослушивать. В данном случае, контейнер будет прослушивать порт `3000` внутри себя, что означает, что приложение в контейнере будет доступно по этому порту внутри кластера Kubernetes.

Теперь создадим сервис для фронтенда `sample-web-deployments.yaml`

```
kind: Service
apiVersion: v1
metadata:
  name: sample-front
  labels:
    app: sample-front
spec:
  ports:
  - port: 3000
    targetPort: 3000
  selector:
    app: sample-front
  type: NodePort
```

Можно заметить, что использовался порт `3000` для работы сервиса, а в ингрес контроллере указывали `80` при его создании, отредактируем ингресс для доступа через 3000 порт.

```
kubectl edit ing/octavia-ingress
```

В нем необходимо изменить:

```
port:
  number: 80
```

На

```
port:
  number: 3000
```

Проверьте фронтед, для этого необходимо проверим какой ip адрес получил ingress, для этого введите команду:

```
kubectl get ing
```

В столбце `ADDRESS` будет указан внешний адрес балансировщика.

Откройте в браузере фронтенд и убедитесь, что он запустился и работает.

> Стоит заметить, что ответ будет происходить только при запросе по доменному имени. Стоит обеспечить доступность домен по DNS или добавить в файл hosts вашей операционной системы.

После удачной проверки работы фронтенда приступим к добавлению бекенда. back-service-pods.yaml:

```
---
kind: Service
apiVersion: v1
metadata:
  name: sample-back
  labels:
    app: sample-back
spec:
  ports:
  - port: 5000
    targetPort: 5000
  selector:
```

```
    app: sample-back
  type: NodePort
```

back-pods.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-back
  labels:
    app: sample-back
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sample-back
  template:
    metadata:
      labels:
        app: sample-back
    spec:
      containers:
      - name: sample-back
        image: registry.gitlab.resds.ru/itt/sample-project:back
        ports:
        - containerPort: 5000
```

Также необходимо отредактировать ingress, в этот раз мы сделаем это с помощью редактирования конфигурационного файла(`ingress.yaml`):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: openstack
    octavia.ingress.kubernetes.io/internal: "false"
  name: test-octavia-ingress
  namespace: default
spec:
  rules:
```

```
  - host: test.com
    http:
      paths:
      - backend:
          service:
            name: sample-back
            port:
              number: 5000
        path: /api
        pathType: Prefix
      - backend:
          service:
            name: sample-back
            port:
              number: 5000
        path: /api/notes
        pathType: Prefix
      - backend:
          service:
            name: sample-front
            port:
              number: 3000
        path: /
        pathType: Prefix
```

и применим изменившеюся конфигурацию:

```
kubectl apply -f ingress.yaml
```

Проверьте работу веб приложения.

Версия #11
Тарабанов Илья Федорович создал 22 апреля 2024 11:19:02
Тарабанов Илья Федорович обновил 16 мая 2024 19:17:24