Практическое пособие по Pods

1. Введение

Что такое Pod?

Pod — это наименьшая единица развертывания в Kubernetes, которая может содержать один или несколько контейнеров. В большинстве случаев используется один контейнер на Pod, но бывают сценарии, когда несколько контейнеров работают вместе (sidecar-контейнеры).

Цель практики

Изучить базовые команды для работы с Pod, научиться создавать манифесты, подключать конфигурационные файлы, хранить секретные данные и настраивать проверки доступности.

---

2. Работа с Pod через kubectl

2.1. Создание Pod через команду

1. Запустите под с помощью команды:

```
kubectl run hello --image=nginx
```

2. Проверьте статус пода:

```
kubectl get pods
```

3. Откройте подробную информацию о поде:

```
kubectl describe pod hello
```

4. Отправьте команду внутрь пода:

```
kubectl exec hello -- date
```

5. Зайдите внутрь пода в интерактивном режиме:

```
kubectl exec -it hello -- sh
```

6. Выйдите из контейнера:

```
exit
```

7. Посмотрите логи пода:

```
kubectl logs hello
```

---

3. Создание Pod через манифест

3.1. Одиночный контейнер

1. Создайте папку проекта:

```
mkdir k8s
cd k8s
```

2. Создайте файл `pod_web.yaml` с содержанием:

```
# Определяем версию API для объекта Kubernetes
apiVersion: v1

# Тип объекта, который мы создаем (Pod — это базовая единица развертывания)
kind: Pod

# Метаданные для идентификации Pod (имя и другая информация)
metadata:
  name: my-web  # Имя пода

# Спецификация пода, описывающая контейнеры и их параметры
spec:
  containers:
    - name: container-apache  # Имя контейнера
      image: php:8.1-apache  # Образ контейнера с PHP и Apache
      ports:
        - containerPort: 80  # Открываем порт 80 внутри контейнера для HTTP-
трафика
```

3. Примените манифест:

```
kubectl apply -f pod_web.yaml
```

4. Проверьте статус:

```
kubectl get pods
```

5. Откройте информацию о поде:

```
kubectl describe pod my-web
```

6. Удалите под:

```
kubectl delete -f pod_web.yaml
```

---

4. Работа с Volume

1. Создайте файл `pod_with_volume.yaml`:

```
# Определяем версию API для объекта Kubernetes
apiVersion: v1

# Тип объекта, который мы создаем (Pod — это базовая единица развертывания)
kind: Pod
```

```
# Метаданные для идентификации Pod (имя пода)
metadata:
  name: pod-volume  # Имя пода

# Спецификация пода, включая описание томов и контейнеров
spec:
  volumes:
    - name: cache  # Имя тома
      emptyDir: {}  # Пустой том, который создается при запуске пода и
удаляется при его удалении
  containers:
    - name: web  # Имя контейнера
      image: nginx  # Образ контейнера с Nginx
      volumeMounts:
        - mountPath: /cache  # Путь в контейнере, куда будет примонтирован
том
          name: cache  # Имя тома, который будет примонтирован
```

2. Запустите под:

```
kubectl apply -f pod_with_volume.yaml
```

3. Проверьте сохранность данных внутри Volume после перезапуска пода.

---

5. ConfigMap и Secret

5.1. ConfigMap

1. Создайте ConfigMap:

```
kubectl create configmap app-config --from-literal=APP_ENV=prod
```

2. Используйте его в манифесте:

```
# Определяем переменную окружения для контейнера
env:
  - name: APP_ENV  # Имя переменной окружения
    valueFrom:
      configMapKeyRef:
        name: app-config  # Имя ConfigMap, из которого получаем значение
        key: APP_ENV  # Ключ в ConfigMap, содержащий значение переменной
# ConfigMap используется для хранения конфигурационных значений, которые
могут быть переиспользованы в нескольких подах.
# Это позволяет централизованно управлять настройками без изменения манифеста
пода.
```

5.2. Secret

1. Создайте Secret:

```
kubectl create secret generic db-secret --from-
literal=password=SuperSecret123
```

2. Используйте его в манифесте:

```
# Определяем переменную окружения для контейнера
env:
  - name: DB_PASSWORD  # Имя переменной окружения для пароля БД
    valueFrom:
      secretKeyRef:
        name: db-secret  # Имя Secret, содержащего чувствительные данные
        key: password  # Ключ, содержащий пароль
# Secret используется для хранения чувствительных данных, таких как пароли
или токены.
# В отличие от ConfigMap, Secret кодируется в base64 и ограничивает доступ к
данным, обеспечивая большую безопасность.
```

---

6. Пробы

1. Настройте проверку доступности контейнера:

```
# Настройка проверки живучести контейнера
livenessProbe:
  httpGet:
    path: /  # Путь, который будет запрашиваться для проверки
работоспособности
    port: 80  # Порт, на котором работает приложение
  initialDelaySeconds: 3  # Время задержки перед первой проверкой после
запуска контейнера
  periodSeconds: 10  # Частота проверок в секундах
# LivenessProbe используется для автоматического перезапуска контейнера, если
он перестает отвечать.
# Это полезно для самовосстановления приложений, которые могут зависнуть или
перестать работать.
```

---

7. Масштабирование

1. Создайте реплику:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: web-replicas
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web
        image: nginx
        ports:
        - containerPort: 80
```

2. Примените манифест:

```
kubectl apply -f replicas.yaml
```

3. Проверьте количество подов:

```
kubectl get pods
```

---

8. Домашнее задание

1. Создайте Pod с двумя контейнерами, которые взаимодействуют через общий Volume.
2. Подключите ConfigMap и Secret.
3. Настройте пробы для проверки доступности контейнеров.
4. Настройте сетевые политики для блокировки доступа к API.

Итог

Вы научились создавать поды, управлять конфигурацией, защищать секретные данные и масштабировать приложение в Kubernetes.

Если остались вопросы, пишите в чат курса или поднимайте руку на занятии!