Kubernetes controllers. ReplicaSet, Deployment, DaemonSet

Подготовка

Домашняя работа предполагает выполнение в локальном кластере kind

Для начала установим kind и создадим кластер. Инструкция по быстрому старту

Будем использовать следующую конфигурацию нашего локального кластера - `kind-config.yaml`:

```
kind: Cluster
apiVersion: kind.sigs.k8s.io/v1alpha3
nodes:
- role: control-plane
- role: control-plane
- role: control-plane
- role: worker
- role: worker
- role: worker
```

Подготовка

Запустите создание кластера kind:

```
kind create cluster --config kind-config.yaml
```

После появления отчета об успешном создании убедитесь что развернуто три master ноды и три worker ноды:

```
$ kubectl get nodes
NAME                  STATUS   ROLES    AGE     VERSION
kind-control-plane    Ready    master   5m16s   v1.16.3
kind-control-plane2   Ready    master   4m14s   v1.16.3
kind-control-plane3   Ready    master   3m3s    v1.16.3
kind-worker           Ready    <none>   2m9s    v1.16.3
kind-worker2          Ready    <none>   2m8s    v1.16.3
kind-worker3          Ready    <none>   2m9s    v1.16.3
```

ReplicaSet

В предыдущем домашнем задании мы запускали standalone pod с микросервисом `frontend`. Пришло время доверить управление pod'ами данного микросервиса одному из контроллеров Kubernetes.

Начнем с ReplicaSet и запустим одну реплику микросервиса `frontend`.

- Создайте и примените манифест `frontend-replicaset.yaml` с содержимым со следующей страницы
- Не забудьте изменить образ на собранный в предущем ДЗ

ReplicaSet

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: server
        image: avtandilko/hipster-frontend:v0.0.1
```

ReplicaSet

- Как можно понять из появившейся ошибки - в описании ReplicaSet не хватает важной секции
- Определите, что необходимо добавить в манифест, исправьте его и примените вновь. Подсказку можно найти в готовом примере манифеста
- Не забудьте про то, что без указания environment переменных сервис не заработает

В результате вывод команды `kubectl get pods -l app=frontend` должен показывать, что запущена одна реплика микросервиса `frontend`:

```
NAME             READY   STATUS    RESTARTS   AGE
frontend-klglk   1/1     Running   0          73s
```

ReplicaSet

Одна работающая реплика - это уже неплохо, но в реальной жизни, как правило, требуется создание нескольких инстансов одного и того же сервиса для:

- Повышения отказоустойчивости
- Распределения нагрузки между репликами

Давайте попробуем увеличить количество реплик сервиса ad-hoc командой:

```
kubectl scale replicaset frontend --replicas=3
```

ReplicaSet

Проверить, что ReplicaSet контроллер теперь управляет тремя репликами, и они готовы к работе, можно следующим образом:

```
$ kubectl get rs frontend
NAME       DESIRED   CURRENT   READY   AGE
frontend   3         3         3       13m
```

Проверим, что благодаря контроллеру pod'ы действительно восстанавливаются после их ручного удаления:

```
kubectl delete pods -l app=frontend | kubectl get pods -l app=frontend -w
```

ReplicaSet

- Повторно примените манифест `frontend-replicaset.yaml`
- Убедитесь, что количество реплик вновь уменьшилось до одной
- Измените манифест таким образом, чтобы из манифеста сразу разворачивалось три реплики сервиса, вновь примените его

Обновление ReplicaSet

Давайте представим, что мы обновили исходный код и хотим выкатить новую версию микросервиса

- Добавьте на DockerHub версию образа с новым тегом (**v0.0.2**, можно просто перетегировать старый образ)
- Обновите в манифесте версию образа
- Примените новый манифест, параллельно запустите отслеживание происходящего:

```
kubectl apply -f frontend-replicaset.yaml | kubectl get pods -l app=frontend -w
```

> Кажется, ничего не произошло

Обновление ReplicaSet

Давайте проверим образ, указанный в ReplicaSet:

```
kubectl get replicaset frontend -o=jsonpath='{.spec.template.spec.containers[0].image}'
```

И образ из которого сейчас запущены pod, управляемые контроллером:

```
kubectl get pods -l app=frontend -o=jsonpath='{.items[0:3].spec.containers[0].image}'
```

Обратите внимание на использование ключа `-o jsonpath` для форматирования вывода. Подробнее с данным функционалом kubectl можно ознакомиться по ссылке

Обновление ReplicaSet

- Удалите все запущенные pod и после их пересоздания еще раз проверьте, из какого образа они развернулись
- Руководствуясь материалами лекции опишите произошедшую ситуацию, почему обновление ReplicaSet не повлекло обновление запущенных pod?
- Мы, тем временем, перейдем к следующему контроллеру, более подходящему для развертывания и обновления приложений внутри Kubernetes

Deployment

Для начала - воспроизведите действия, проделанные с микросервисом `frontend` для микросервиса `paymentService`.

Результат:

- Собранный и помещенный в Docker Hub образ с двумя тегами **v0.0.1** и **v0.0.2**
- Валидный манифест `paymentservice-replicaset.yaml` с тремя репликами, разворачивающими из образа версии **v0.0.1**

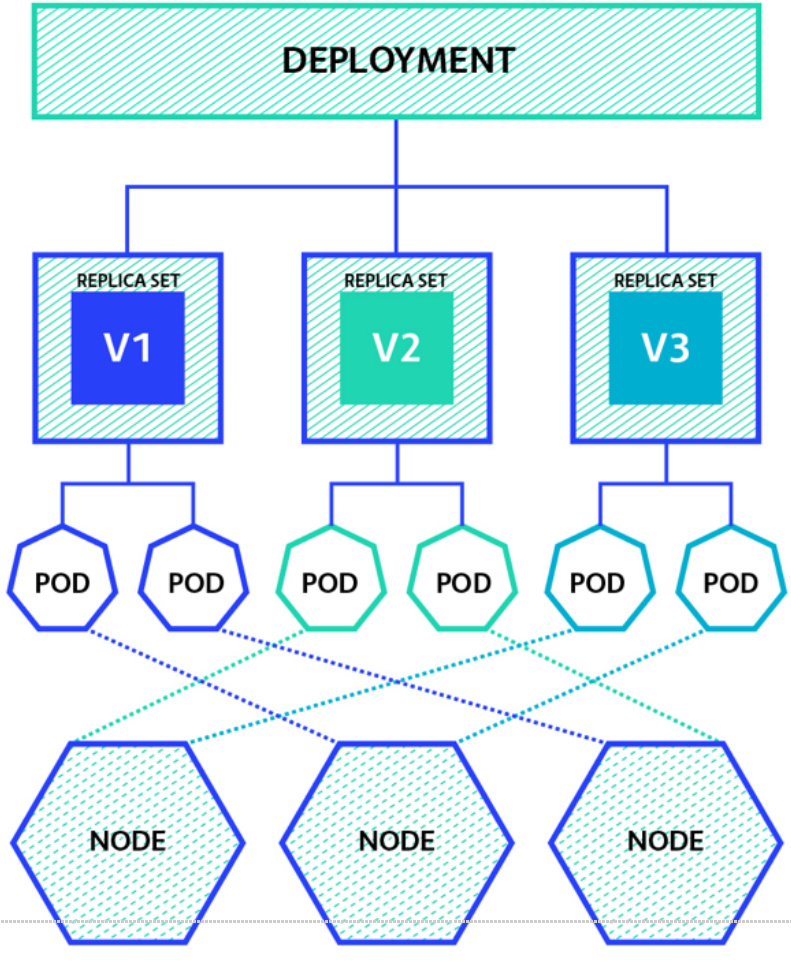
Deployment

Приступим к написанию Deployment манифеста для сервиса `payment`

- Скопируйте содержимое файла `paymentservice-replicaset.yaml` в файл `paymentservice-deployment.yaml`
- Измените поле `kind` с **ReplicaSet** на **Deployment**
- Манифест готов 😉 Примените его и убедитесь, что в кластере Kubernetes действительно запустилось три реплики сервиса `payment` и каждая из них находится в состоянии `Ready`
- Обратите внимание, что помимо Deployment (`kubectl get deployments`) и трех pod, у нас появился новый ReplicaSet (`kubectl get rs`)

Deployment

Вспомним зависимость между Deployment, ReplicaSet и Pod:



Обновление Deployment

Давайте попробуем обновить наш Deployment на версию образа v0.0.2

```
kubectl apply -f paymentservice-deployment.yaml | kubectl get pods -l app=paymentservice
-w
```

Обратите внимание на последовательность обновления pod. По умолчанию применяется стратегия Rolling Update:

- Создание одного нового pod с версией образа **v0.0.2**
- Удаление одного из старых pod
- Создание еще одного нового pod
- ...

Обновление Deployment

Убедитесь что:

- Все новые pod развернуты из образа **v0.0.2**
- Создано два ReplicaSet:
  - Один (новый) управляет тремя репликами pod с образом **v0.0.2**
  - Второй (старый) управляет нулем реплик pod с образом **v0.0.1**

Также мы можем посмотреть на историю версий нашего Deployment:

```
kubectl rollout history deployment paymentservice
```

Deployment | Rollback

Представим, что обновление по каким-то причинам произошло неудачно и нам необходимо сделать откат. Kubernetes предоставляет такую возможность:

```
kubectl rollout undo deployment paymentservice --to-revision=1 | kubectl get rs -l
app=paymentservice -w
```

В выводе мы можем наблюдать, как происходит постепенное масштабирование вниз "нового" ReplicaSet, и масштабирование вверх "старого"

Deployment | Задание со ⭐

С использованием параметров `maxSurge` и `maxUnavailable` самостоятельно реализуйте два следующих сценария развертывания:

- **Аналог blue-green**:
    1. Развертывание трех новых pod
    2. Удаление трех старых pod
- **Reverse Rolling Update**:
    1. Удаление одного старого pod
    2. Создание одного нового pod
    3. ...

Deployment | Задание со ⭐

Документация с описанием стратегий развертывания для Deployment.

В результате должно получиться два манифеста:

- `paymentservice-deployment-bg.yaml`
- `paymentservice-deployment-reverse.yaml`

Probes

Мы научились разворачивать и обновлять наши микросервисы, но можем ли быть уверены, что они корректно работают после выкатки? Один из механизмов Kubernetes, позволяющий нам проверить это - Probes

Давайте на примере микросервиса `frontend` посмотрим на то, как probes влияют на процесс развертывания.

- Создайте манифест `frontend-deployment.yaml` из которого можно развернуть три реплики pod с тегом образа **v0.0.1**
- Добавьте туда описание readinessProbe. Описание можно взять из манифеста по ссылке

Probes

Примените манифест с readinessProbe. Если все сделано правильно, то мы вновь увидим три запущенных pod в описании которых (`kubectl describe pod`) будет указание на наличие readinessProbe и ее параметры

Давайте попробуем сымитировать некорректную работу приложения и посмотрим, как будет вести себя обновление:

- Замените в описании пробы URL `/_healthz` на `/_health`
- Разверните версию **v0.0.2**

> В манифесте, который попадет в PR, readinessProbe должна остаться рабочей

Probes

Если посмотреть на текущее состояние нашего микросервиса, мы увидим, что был создан один pod новой версии, но его статус готовности следующий:

```
NAME                                READY   STATUS    RESTARTS   AGE
frontend-6bf67c4974-2cns9           0/1     Running   0          10s
```

Команда `kubectl describe pod` поможет нам понять причину:

```
Events:
  Type     Reason     Age                     From                  Message
  ----     ------     ----                    ----                  -------
  Warning  Unhealthy  11s (x22 over 3m41s)  kubelet, kind-worker  Readiness probe
failed: HTTP probe failed with statuscode: 404
```

Probes

Как можно было заметить, пока readinessProbe для нового pod не станет успешной - Deployment не будет пытаться продолжить обновление.

На данном этапе может возникнуть вопрос - как автоматически отследить успешность выполнения Deployment (например для запуска в CI/CD).

В этом нам может помочь следующая команда:

```
kubectl rollout status deployment/frontend
```

Probes

Таким образом описание pipeline, включающее в себя шаг развертывания и шаг отката, в самом простом случае может выглядеть так (синтаксис GitLab CI):

```
deploy_job:
  stage: deploy
  script:
    - kubectl apply -f frontend-deployment.yaml
    - kubectl rollout status deployment/frontend --timeout=60s

rollback_deploy_job:
  stage: rollback
  script:
    - kubectl rollout undo deployment/frontend
  when: on_failure
```

DaemonSet

Рассмотрим еще один контроллер Kubernetes. Отличительная особенность DaemonSet в том, что при его применении на каждом физическом хосте создается по одному экземпляру pod, описанного в спецификации.

Типичные кейсы использования DaemonSet:

- Сетевые плагины
- Утилиты для сбора и отправки логов (Fluent Bit, Fluentd, etc...)
- Различные утилиты для мониторинга (Node Exporter, etc...)
- ...

DaemonSet | Задание со ⭐

Опробуем DaemonSet на примере Node Exporter

- Найдите в интернете или напишите самостоятельно манифест `node-exporter-daemonset.yaml` для развертывания DaemonSet с Node Exporter
- После применения данного DaemonSet и выполнения команды: `kubectl port-forward <имя любого pod в DaemonSet> 9100:9100` метрики должны быть доступны на localhost: `curl localhost:9100/metrics`

DaemonSet | Задание с ⭐⭐

- Как правило, мониторинг требуется не только для worker, но и для master нод. При этом, по умолчанию, pod управляемые DaemonSet на master нодах не разворачиваются
- Найдите способ модернизировать свой DaemonSet таким образом, чтобы Node Exporter был развернут как на master, так и на worker нодах (конфигурацию самих нод изменять нельзя)
- Отразите изменения в манифесте

Проверка ДЗ

Основная часть домашнего задания проверяется автоматически. После успешного прохождения тестов вы можете самостоятельно сделать Merge в master ветку.

Если вы сделали одно или несколько заданий со ⭐ и хотите получить комментарии преподавателя:

- Добавьте к PR метку `Review Required`
- Не делайте Merge самостоятельно
- Тесты основной части задания должны быть успешными

Проверка ДЗ

Поместите все файлы, созданные в процессе выполнения домашнего задания в директорию kubernetes-controllers

Структура репозитория при выполнении всех заданий со ⭐:

```
├── kubernetes-intro
└── kubernetes-controllers
    ├── frontend-deployment.yaml
    ├── frontend-replicaset.yaml
    ├── node-exporter-daemonset.yaml
    ├── paymentservice-deployment-bg.yaml
    ├── paymentservice-deployment-reverse.yaml
    ├── paymentservice-deployment.yaml
    └── paymentservice-replicaset.yaml
```

Проверка ДЗ

- Результаты вашей работы должны быть добавлены в ветку **kubernetes-controllers** вашего GitHub репозитория `<YOUR_LOGIN>_platform`
- В **README.md** рекомендуется внести описание того, что сделано
- Создайте Pull Request к ветке **master** (описание PR рекомендуется заполнять)
- Добавьте метку `kubernetes-controllers` к вашему PR

Проверка ДЗ

- После того как автоматизированные тесты проверят корректность выполнения ДЗ, необходимо сделать merge ветки **kubernetes-controllers** в `master` и закрыть PR
- Если у вас возникли вопросы по ДЗ и необходима консультация преподавателей - после прохождения автотестов добавьте к PR метку `Review Required` и **не мерджите PR** самостоятельно