KUB 11: Базовые ресурсы в Kubernetes

Описание:

Итак, за последние 10 заданий мы с вами познакомились с общей архитектурой Kubernetes. Изучили компоненты control plane, worker nodes, а так же рассмотрели API, который связывает все компоненты между собой. Помимо этого мы увидели, что API разбит на группы, внутри которых содержатся разные ресурсы. Собственно конкретно с этого задания мы начнем активно обсуждать ресурсы, которые содержатся в Kubernetes.

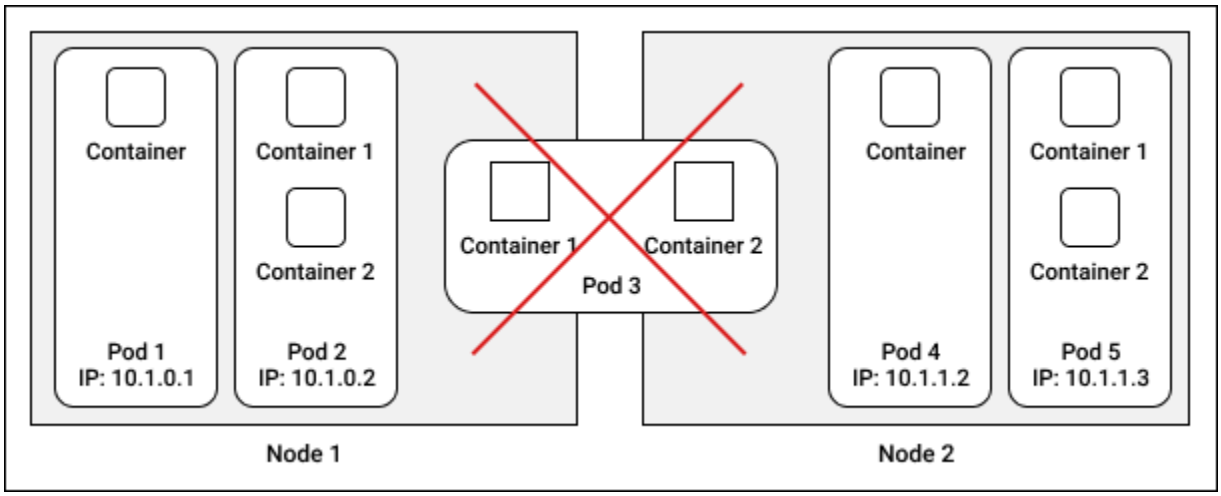
Pod
Kubernetes имеет вполне богатую модель объектов. Эти объекты представляют собой приложения, запросы на ресурсы, политики, правила и многое другое. С описанием каждого объекта мы сообщаем кластеру требуемое состояние посредством описания спецификации. A controller-manager, в свою очередь, начинает изменения (создание/удаление ресурсов) в тех случаях, когда желаемое состояние не соответствует фактическому, непрерывно наблюдая за состоянием кластера, сравнивая результат с ожидаемым. Примеры объектов: Pod, ReplicaSet, Deployment, Namespace. Обычно спецификация объектов описывается в YAML, но можно послать JSON API-серверу. Помните, мы делали pod, используя команду `kubectl run`? Давайте посмотрим, как выглядело бы его описание в yaml:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
```

```
  name: nginx
```

Первая строка `apiVersion` задает требуемый API для обработки данного объекта. Вспомните структуру API, которую отдавал API-сервер. Здесь может быть указана группа плюс версия API. `Kind` — задает тип объекта в правильной группе API, а `metadata` — добавляет информацию к этому объекту, например, имя. Внутри `spec` мы описываем последнее состояние объекта — что мы хотим получить внутри. В этом случае описание у нас вполне простое — мы хотим запустить один контейнер внутри pod с образом nginx. Как мы говорили до этого в одном поде контейнеров бывает несколько — достаточно добавить их описание в yaml-файл.

Pod представляет собой группу из одного или нескольких контейнеров. Это неделимая единица, с точки зрения Kubernetes. Они всегда выполняются на одном рабочем узле, в одном пространстве имени Linux. Каждый Pod подобен машине с IP-адресом, сетью, процессами, дисками. Приложение может состоять из нескольких процессов, например, front + back. Для безошибочной работы приложения вам, скорее всего, нужно будет запускать эти процессы в единой экосистеме. Каждый процесс будет запущен в отдельном контейнере. Набор контейнеров внутри Pod неделим. Если в составе pod находятся 4 контейнера, нельзя запустить два на одной физической машине и два — на другой.

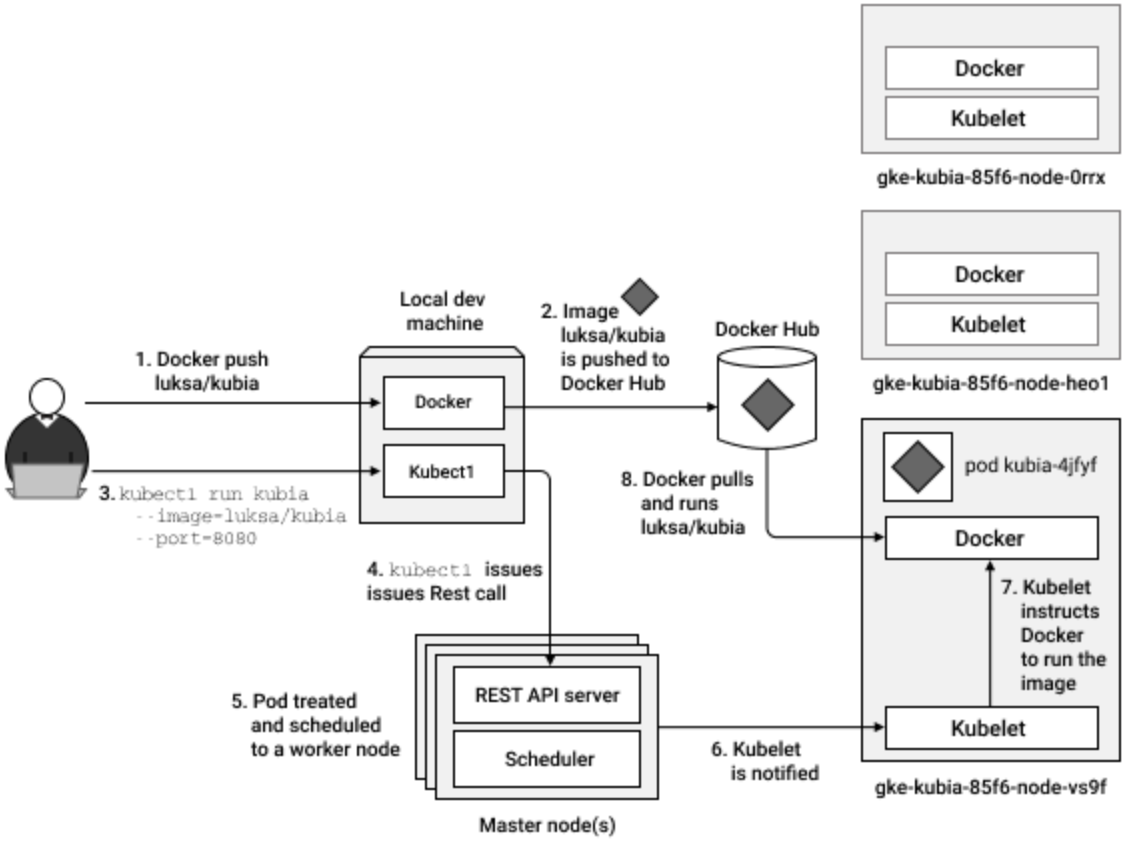


Давайте изучим, как может выглядеть спецификация запуска нескольких контейнеров в одном поде:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
  - image: ubuntu
    name: ubuntu
    command: ["/bin/bash"]
    args: ["-c", "while true; do sleep 300; done"]
```

Таким образом вы можете запустить в поде несколько контейнеров для дополнительных задач.

При создании пода мы применили его описание из yaml-файла, которое было конвертировано в json и отправлено к API сервера. Давайте разберемся, что произошло в этот момент:



После отправки REST-запроса controller-manager определил различия между желаемым и фактическим состоянием кластера, передав на исполнение Kubernetes через API информацию о создании требуемого ресурса типа Pod. Далее scheduler назначил положение пода между узлами кластера, и информация об этом была передана kubelet на целевой ноде. В случае с minikube выбора, в общем-то, нет, но это упрощенная схема. Kubelet с помощью CRI передал информацию о необходимости создания контейнера в container runtime, реализованный с помощью runc и containerd (в нашем случае), после чего был создан требуемый ресурс. Именно это имеется в виду, когда произносится упрощенная фраза «Kubernetes создал под».

Namespaces

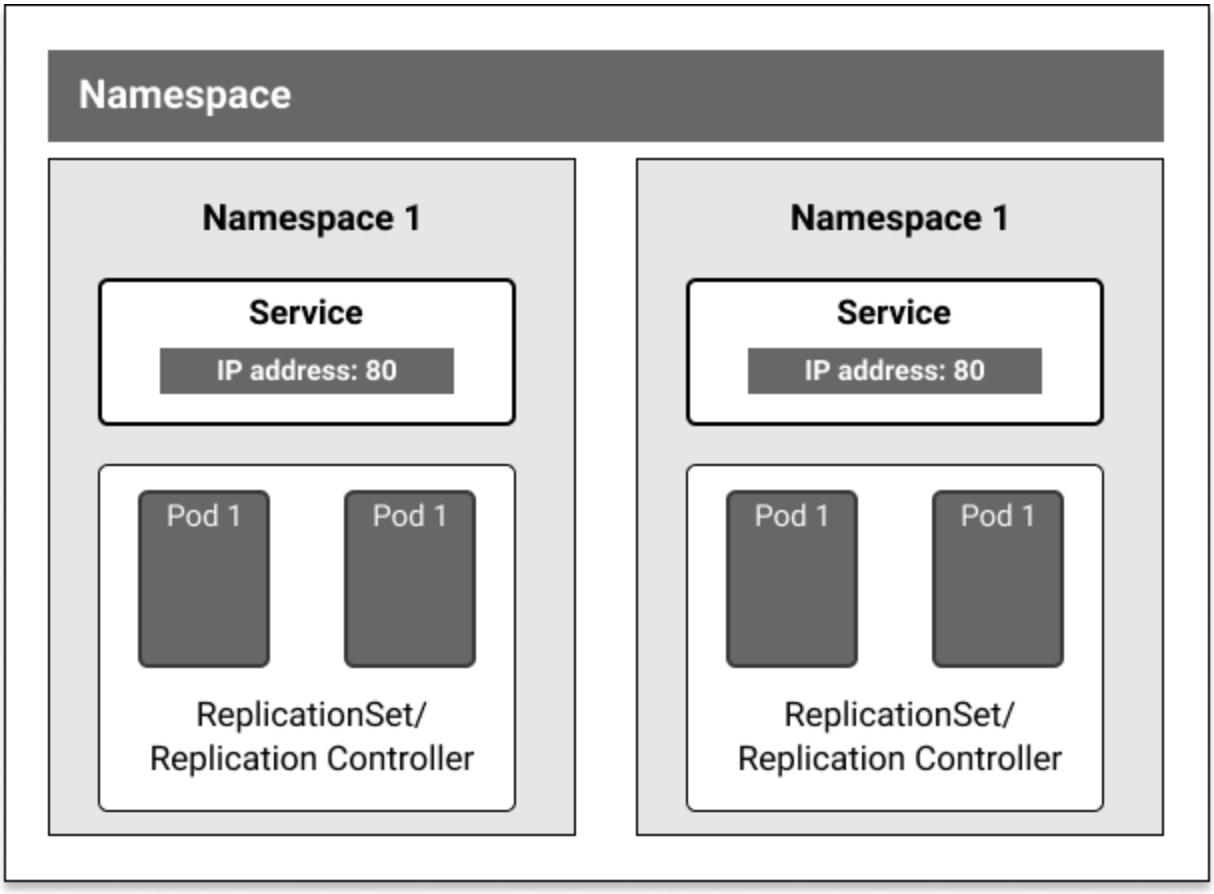
Если кластером пользуется множество людей, то есть смысл разбить их на группы. Для этого, а также для решения многих других задач, в Kubernetes существует абстракция namespaces. Условно можно представить namespace как

частичку от кластера, выделенную для работы определенным людям или процессам.

Внутри namespace можно создавать разные объекты, такие как Pod, Deployment, Service и т.д. Имя должно быть уникальным в рамках одного namespace, но может повторяться в других namespaces. По умолчанию создается два namespaces: `kube-system` и `default`. `Kube-system` содержит системные компоненты кластера, а `default` — объекты без namespace. `kubectl` по умолчанию подключается к namespace default.

Namespaces являются центральной частью многих функций кластера Kubernetes. В рамках `namespace` можно организовать политики безопасности, такие как RBAC и PSP (коснемся далее), сами `namespaces` можно ограничивать по ресурсам и многое другое. Базисным подходом в организации namespaces является организация окружений по назначению — к примеру, `prod` окружение в соответствующем `namespace`, которому выделено максимальное количество ресурсов, `dev/stage` — окружения, ограниченные по лимиту ресурсов и со своими политиками доступа и безопасности, динамически создаваемые namespaces под тестирование отдельных решений в своих branches и так далее.

Однако, чтобы научиться работать со сложным, начнем с простого, создания и организации namespaces.

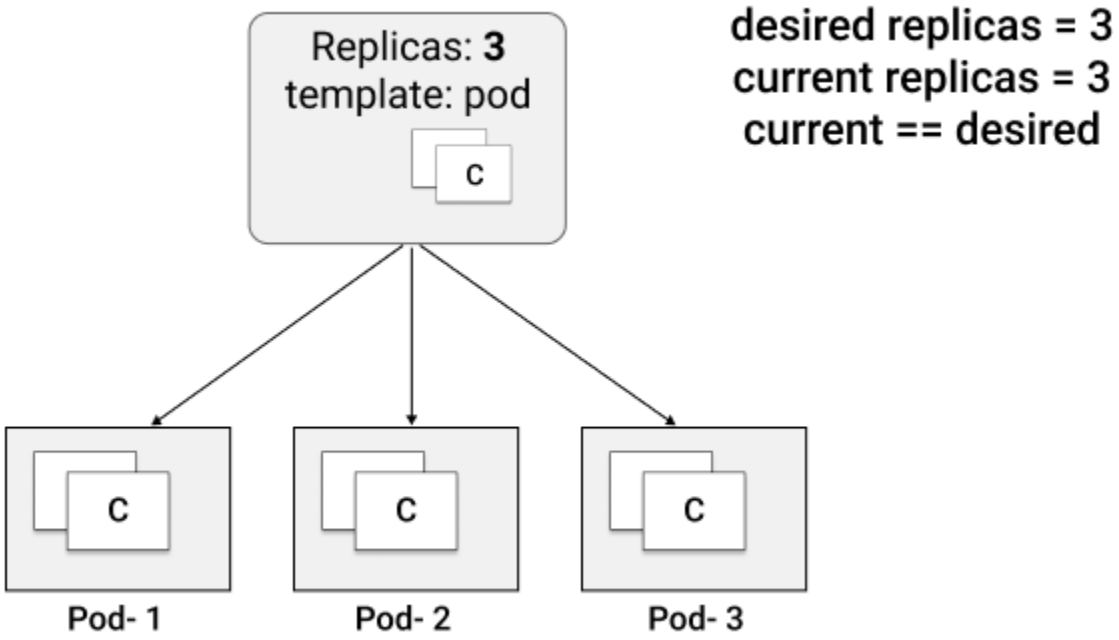


Для создания namespace можно создать yaml-файл с его описанием или выполнить команду `kubectl create namespace test` — так мы создадим namespace `test`. Чтобы создавать объекты внутри нужного нам namespace, мы должны указывать kubectl опцию `-n <namespace>`, которая будет использовать указанный namespace для создания объектов.

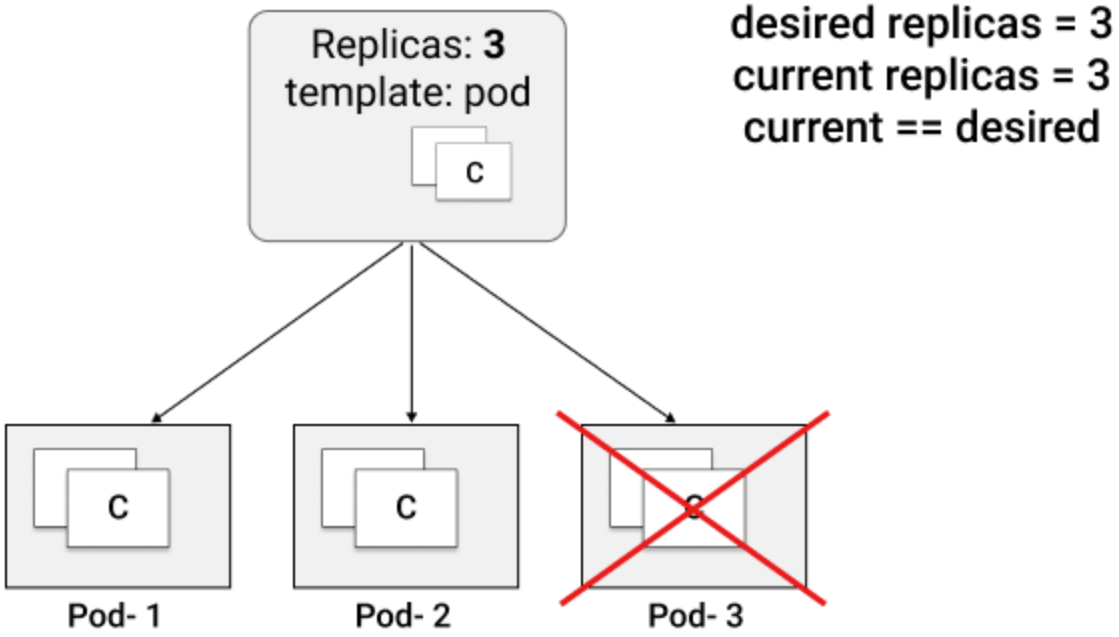
Replica Set

Контроллер репликации (сокращенно `rc`) — часть controller manager мастер-узла. Он следит за соответствием текущего количества pod заданному в spec-директиве. Если количество pod превышает заданное значение, rc убьет лишнее. И наоборот, если количество pod меньше заданного, то rc создаст необходимое. Практически никогда pod не создается без rc, так как за таким pod никто не будет следить, проверять его состояние и перезапускать в случае проблем.

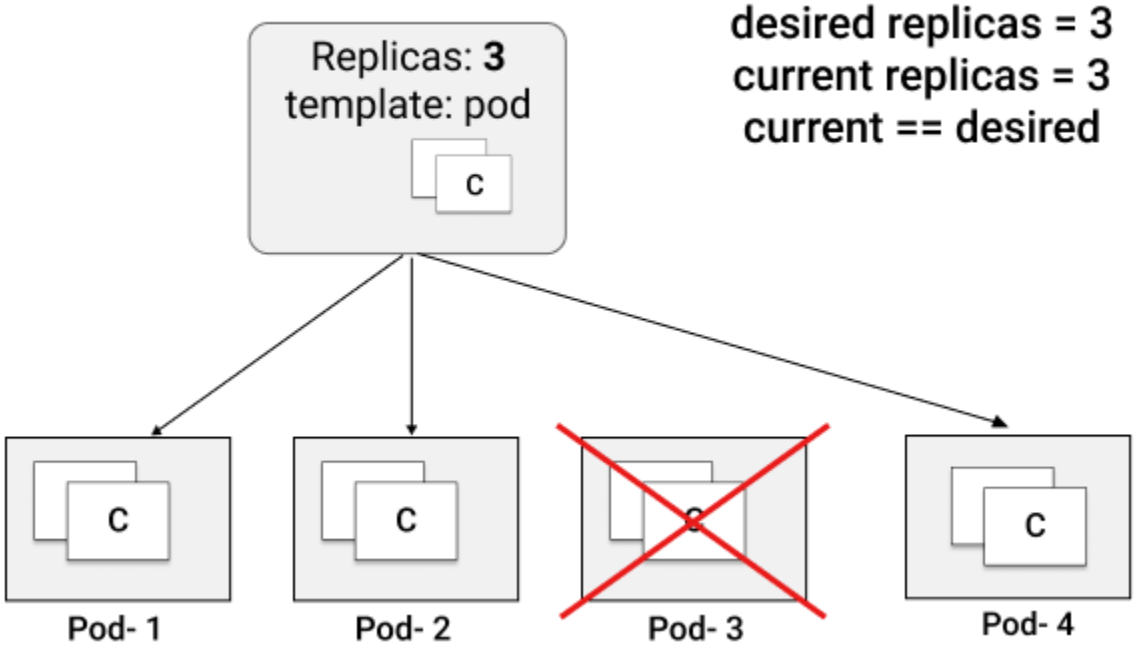
ReplicaSet — это следующее поколение ReplicationController. ReplicaSet поддерживает выборки по меткам на основе равенства и множеств, тогда как ReplicationController — только на основе равенства.



ReplicaSet точно также следит за количеством Pod, перезапуская их в случае сбоя или добавляя/удаляя необходимые.

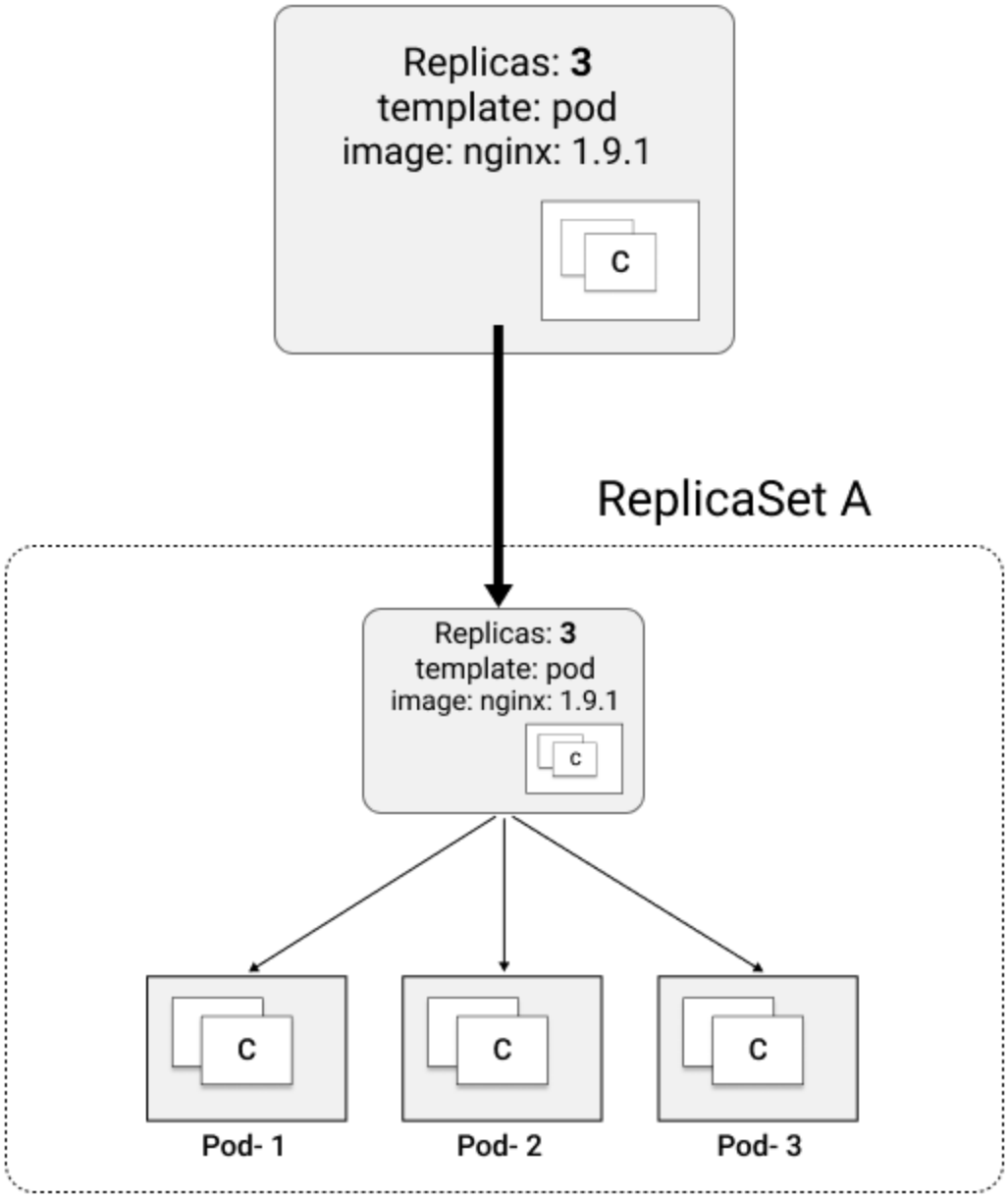


Например, при сбое в работе одного Pod ReplicaSet заменит его новой копией. Текущее состояние не соответствует требуемому.



ReplicaSet может использоваться без Deployment, однако в таком случае обновление конфигурации нужно производить редактированием ReplicaSet. Deployment автоматически создает ReplicaSet, следит за его состоянием и обновляет конфигурацию.

Deployment
Deployment позволяет декларативно описывать конфигурацию нижеследующих ReplicaSet и Pod и применять к ним обновления. В примере Deployment создает ReplicaSet A, ReplicaSet A создает 3 Pod.



Давайте попробуем создать deployment из следующего yaml-файла:

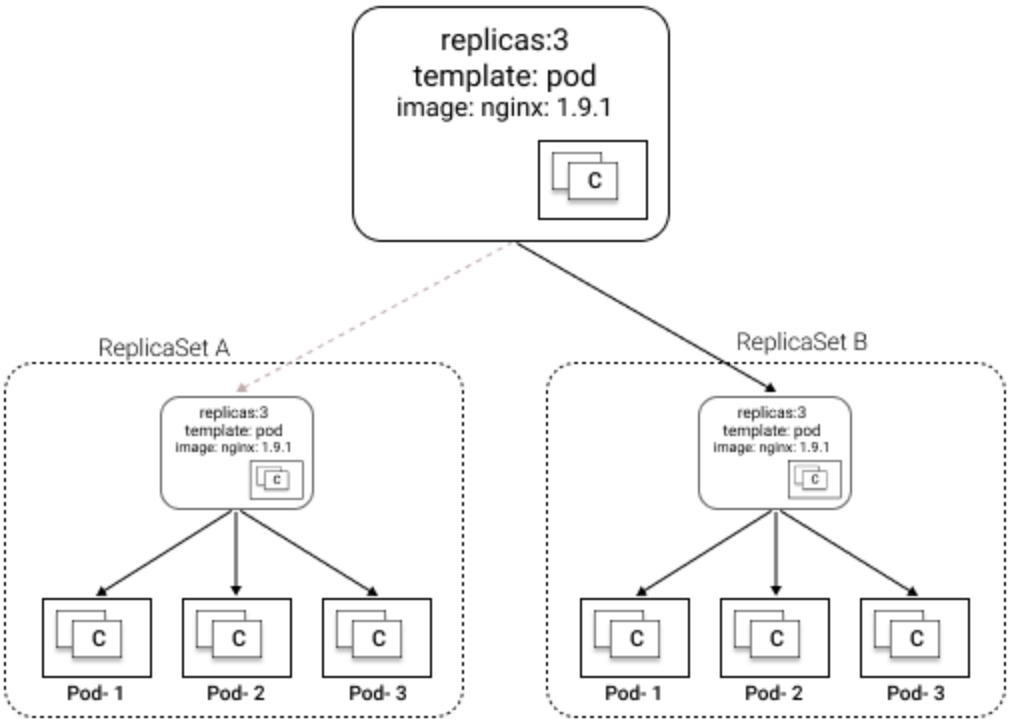
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
```

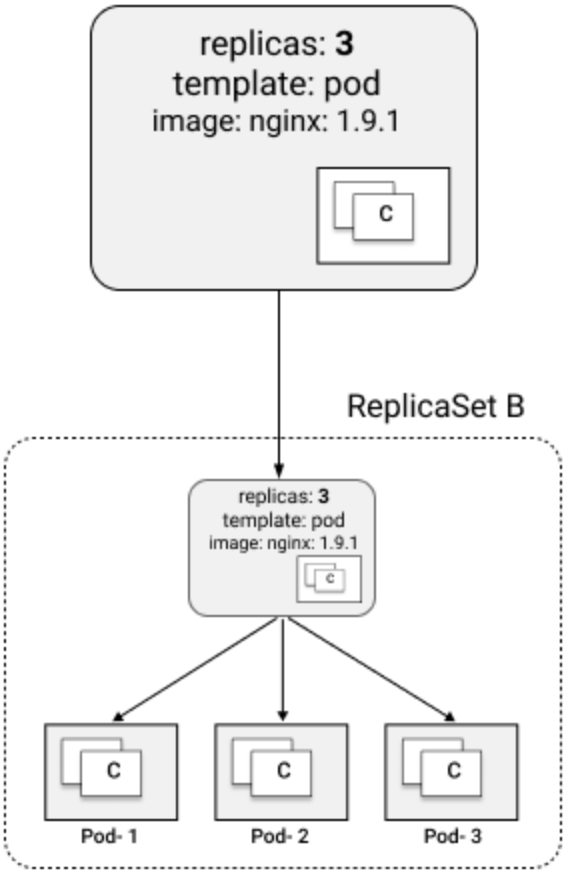
Все достаточно похоже на спецификацию для пода, но есть понятные различия. Во-первых, поменялась `apiVersion` — теперь она показывает на группу apps и на версию v1. Тип ресурса у нас выбран Deployment. Внутри spec указываем количество реплик нашего пода, который описывается в `spec.template`. По сути, это шаблон, по которому Deployment будет создавать поды. Важным моментом здесь является `spec.selectors`, по которому deployment будет искать созданные им поды. Вы можете заметить, что в шаблоне пода также указан label `app=nginx`.

Теперь, когда мы научились делать Deployment, давайте обновим его. Попробуем изменить версию контейнера на nginx:1.9.1. Давайте изменим deployment.yaml файл и применим его заново — `kubectl apply -f deployment.yaml`.

При применении новой конфигурации текущее состояние deployment уже не будет соответствовать запрошенному состоянию. Контроллер Deployment создаст новый ReplicaSet B, который сделает новые Pods.



Как только ReplicaSet B будет готов, Deployment удалит старую ReplicaSet A и будет указывать на новую ReplicaSet B.



Находясь над сущностями ReplicaSet и Pod, Deployment предоставляет возможность обновления и отката. Это возможно, благодаря истории изменения Deployment, которая сохраняется в контроллере.

Также нужно понимать, как можно управлять политикой апгрейда — мы рассмотрим тип апгрейда recreate & rollingUpdate. В yaml это выглядит так:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-dp
spec:
  replicas: 4
  strategy:
```

```
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: redis
```

В этом случае мы указываем, что стратегия апгрейда — RollingUpdate, максимальное число недоступных подов — 0, то есть, всегда сначала будет создаваться новый под, а потом удаляться старый, а также максимальное число подов свыше replicas не должно превышать одного.

Второй типа апгрейда — Recreate. Стратегия просто удаляет все поды и создает их заново.

Полезные ссылки:

- Pods (official docs)
- Overview of Pods
- kubernetes namespaces
- Overview of a Replication Controller
- ReplicaSet (official docs)
- Знакомство с Kubernetes. Часть 4: Реплики (ReplicaSet)

Задание:

1. Создайте namespace `stage`.
2. Внутри namespace `stage` создайте pod `redis-kv` с двумя контейнерами `redis` и `redis-exporter`.
3. Добавьте поду label `app=redis`.
4. Создайте namespace `dev`.
5. Внутри namespace `dev` создайте deployment с именем `web-dp` с одним контейнером внутри — `httpd`.
6. Увеличьте количество реплик `web-dp` до 4.
7. Отправьте задание на проверку.