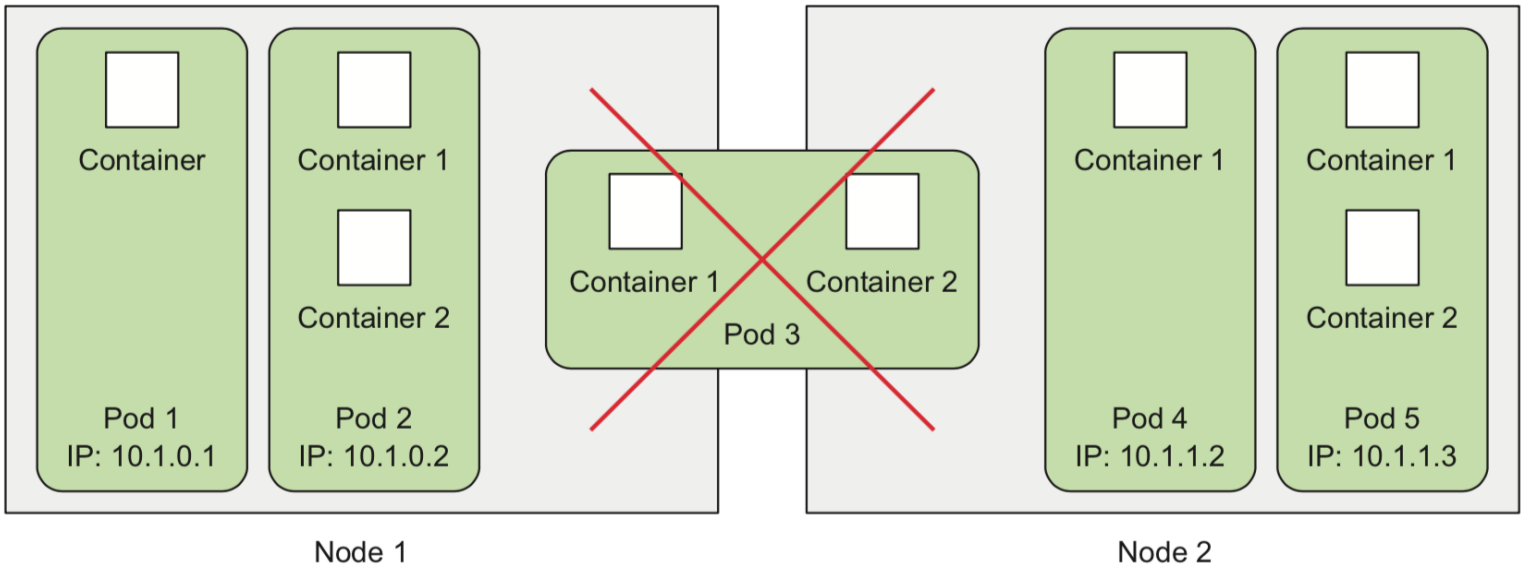
Объекты Kubernetes

Pods

- Группа контейнеров (один или несколько)
- Минимальная сущность, управляемая Kubernetes

  У всех контейнеров общие Network, IPC, UTS, PID* namespaces

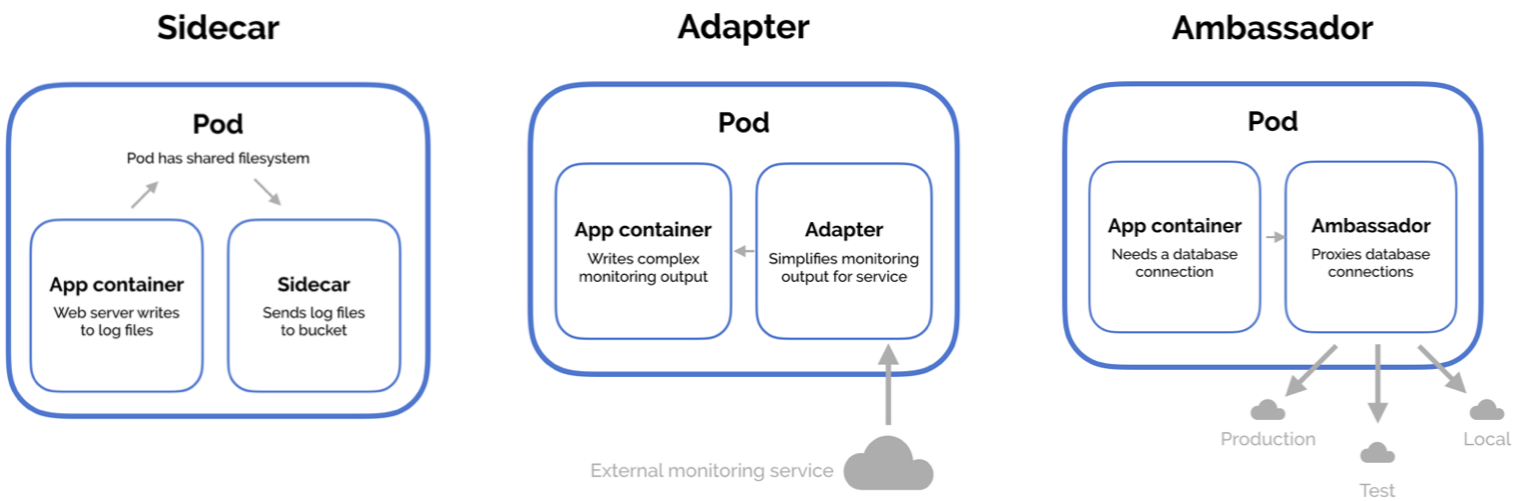


Pods

Контейнеры внутри одного Pod или разные Pod?

- Сервисы должны масштабироваться совместно или по отдельности?
- Должны ли сервисы быть запущены вместе или могут быть разнесены на разные хосты?
- Это связанные сервисы или независимые компоненты?

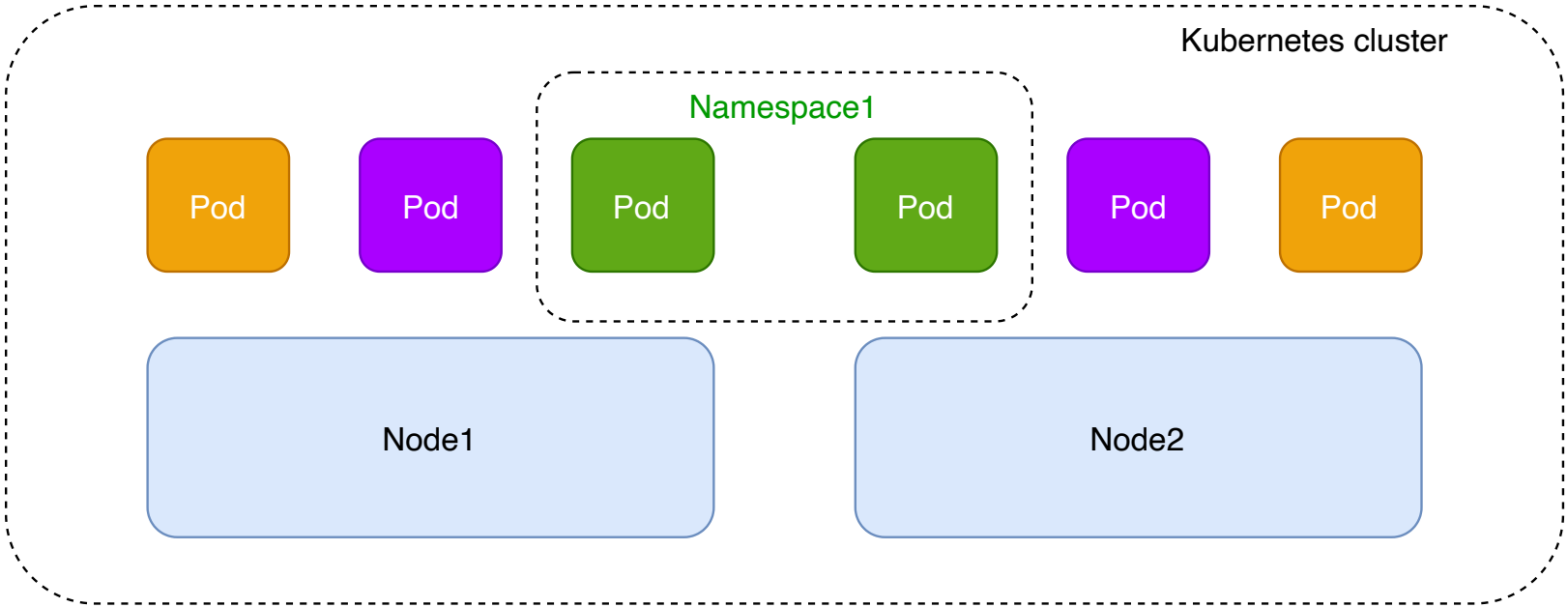
Pods



Описание паттернов

Namespaces

- Можно создать несколько виртуальных кластеров в рамках одного физического кластера
- Namespace - один виртуальный кластер (одно окружение)



Namespaces

У каждой команды есть возможность независимо от других использовать кластер. При этом у команды есть свои:

- Ресурсы (pod, service и т.д.)
- Политики (кому и какие действия можно совершать в кластере)
- Ограничения и квоты на ресурсы

Namespaces

По умолчанию в кластере создается три namespace:

- **default** - для объектов у которых явно не определена принадлежность к другому namespace
- **kube-system** - для системных объектов Kubernetes
- **kube-public** - для объектов к которым нужен доступ из любой точки кластера

Описание объектов

YAML

- 1 файл может описывать много сущностей
- Обязательные поля:
  - apiVersion
  - kind
  - metadata
- Спецификация объекта

Описание объектов

prometheus-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
spec:
  containers:
  - name: prometheus
    image: prom/prometheus:v2.10.0
```

Описание объектов

kind

Тип объекта, который хотим создать

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
spec:
  containers:
  - name: prometheus
    image: prom/prometheus:v2.10.0
```

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: monitoring
```

Описание объектов

apiVersion

Путь на используемую группу API для создания объекта

```
1   apiVersion: $GROUP_NAME/$VERSION
```

3 стадии развития:

- **alpha** - стадия тестирования, функционал может быть удален из последующих версий (v1alpha1)
- **beta** - безопасно для использования, но функционал может меняться (v1beta1)
- **stable** - стабильно (v1)

Документация для версии 1.18

Описание объектов

apiVersion

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: prometheus
spec:
  containers:
  - name: prometheus
    image: prom/prometheus:v2.10.0
```

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob
spec:
-- omitted --
```

Описание объектов

metadata

- Имя создаваемого объекта
- Метки (labels)
- Namespace
- Аннотации

```
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: prometheus
5     labels:
6       app: prometheus
7     namespace: monitoring
8   spec:
9   -- omitted --
```

Полезные команды

```
# Создать объект из манифеста
kubectl create -f manifest.yml
kubectl apply -f manifest.yml
kubectl apply -f link
kubectl apply -f directory/

# Получить список объектов
kubectl get pods

# Получить описание объекта
kubectl describe pod POD_NAME

# Создать объект в ad-hoc режиме
kubectl create namespace NS_NAME
kubectl create ns NS_NAME

# Создать объект в определенном namespace
kubectl create -f manifest.yml -n NS_NAME

# Зайти внутрь контейнера в pod (аналог docker exec)
kubectl exec -ti <POD_NAME> <COMMAND>
```

Практика

Напишите манифест для создания pod **ui** из ранее собранного Docker образа. Pod должен быть развернут в namespace **reddit-entry**. Манифесты можете сохранить в директории `~/project/k8s-objects`

После применения манифеста должен получиться следующий вывод:

```
kubectl get pods -n reddit-entry

NAME                                    READY     STATUS    RESTARTS   AGE
ui                                      1/1       Running   0          1s
```

> Для того, чтобы создать что-либо в namespace, надо создать сам namespace.

Практика

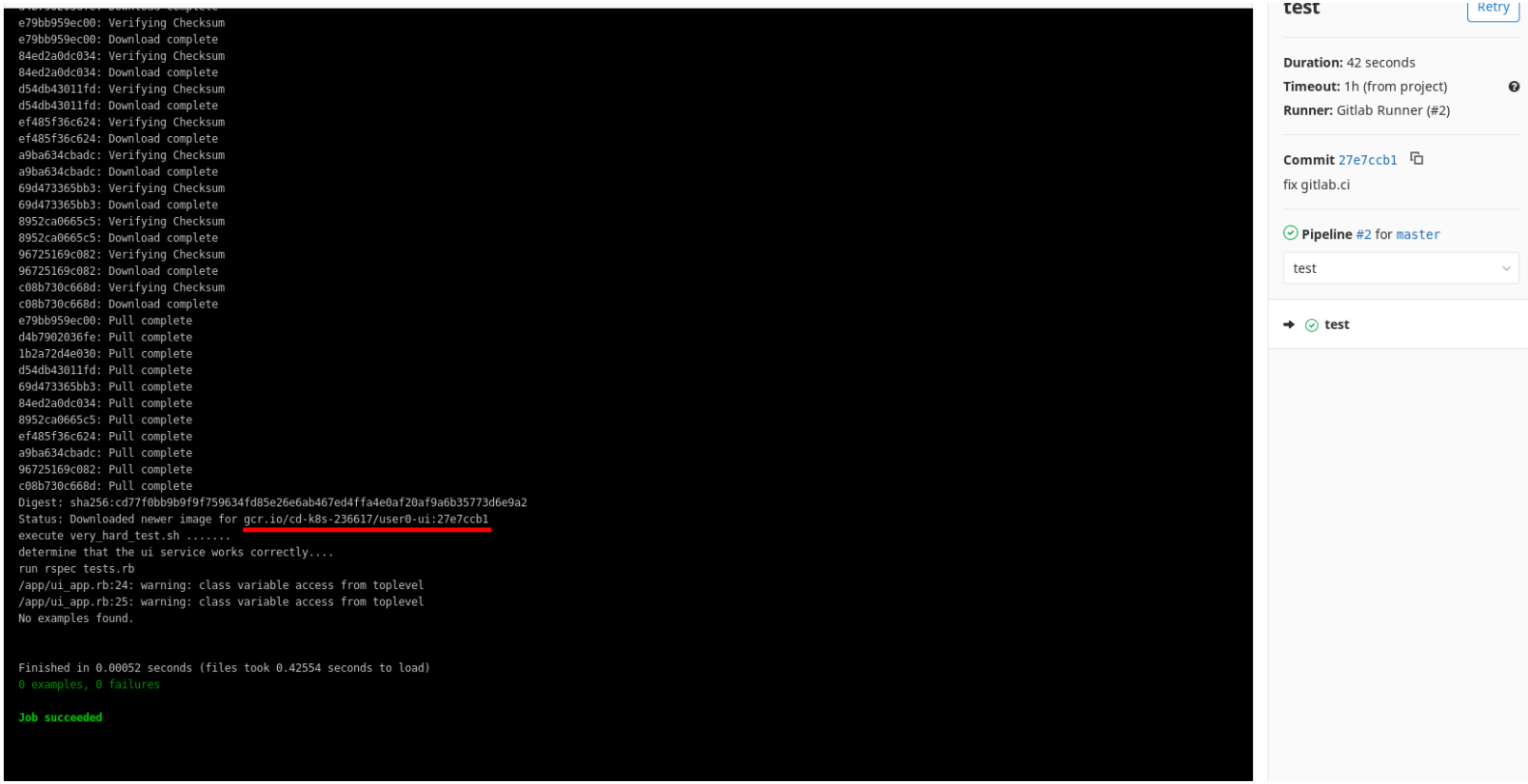
Создадим namespace reddit-entry

```yaml
apiVersion: v1          # Укажем версию API
kind: Namespace         # Тип kind
metadata:
  name: reddit-entry    # Название namespace
```

Сохраним его как reddit-entry.yml и применим `kubectl apply -f reddit-entry.yml`

Практика

Название контейнера можно посмотреть в логе GitLab CI



Практика

Теперь создадим pod, в качестве kind необходимо установить Pod и указать название данного манифеста

```
apiVersion: v1
kind: Pod
metadata:
  name: ui
...
```

Затем необходимо указать название ранее созданного docker контейнера и его имя в k8s.

```
...
spec:
  containers:
  - name: ui
    image: <container name>
```

Практика

Сохраним файл как ui-pod.yml и применим его `kubectl apply -f ui-pod.yml -n reddit-entry`

Resources | Requests

Ресурсы, необходимые для работы контейнера

- Суммарное значение requests всех pod на хосте не может быть большим, чем количество физических ресурсов хоста
- При определении на какой хост поместить тот или иной pod, kube-scheduler будет учитывать значения requests
- Если на хосте недостаточно физических ресурсов для запуска pod, то kube-scheduler гарантированно не выберет данный хост

Resources | Limits

Пороговые значения ресурсов, которые может потреблять контейнер

- При превышении данных значений контейнер будет:
  - Перезапущен в случае превышения по RAM
  - Ограничен в потреблении в случае превышения по CPU
- Суммарное значение limits всех pod на хосте может быть большим, чем количество физических ресурсов хоста (overcommit)

> Указание requests и limits для контейнеров - рекомендуемая практика работы с Kubernetes

Resources | Пример

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.17
    resources:
      requests:
        memory: "64Mi"
        cpu: "100m"
      limits:
        memory: "128Mi"
        cpu: "200m"
```

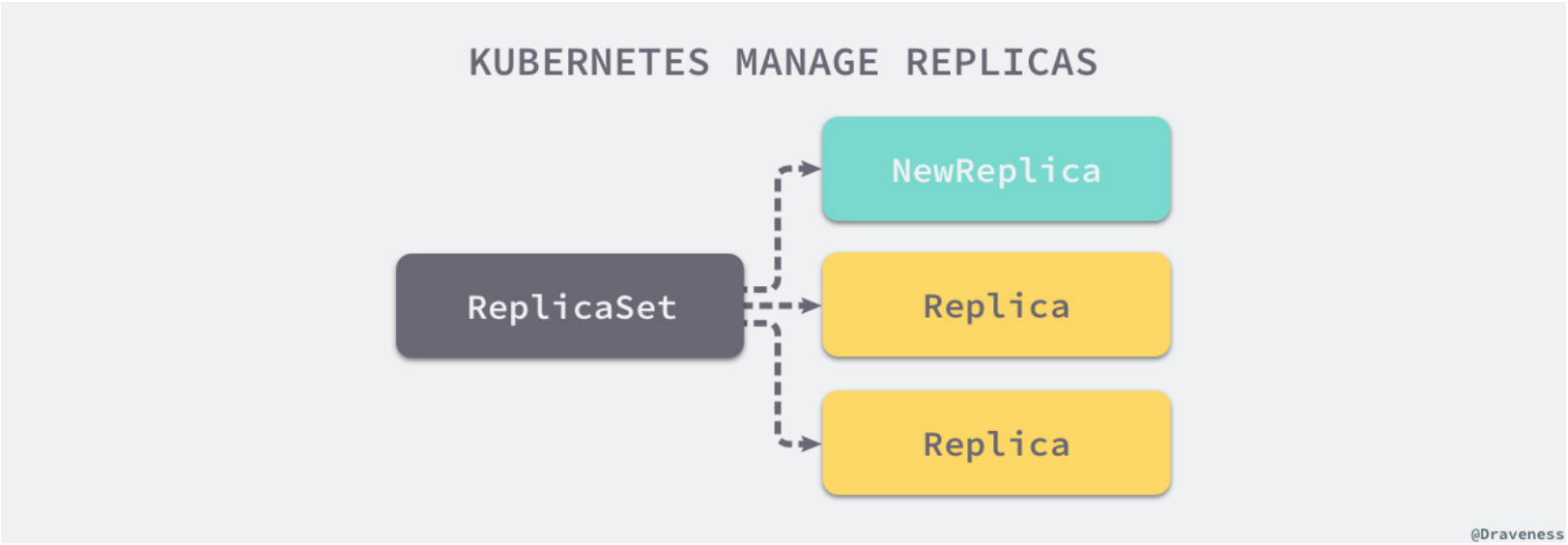
Environment variables

Указание environment переменных внутри контейнера

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.15.8
    env:
    - name: ENV_NAME
      value: ENV_VALUE
```

ReplicaSets

- Контроль заданного количества pod
- Pod, которые будут находиться под управлением rs, выбираются на основании меток



ReplicaSets

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.8
```

Практика

1. Удалите ранее созданный **pod** ui. Сделать это можно командной `kubectl delete pod ui -n reddit-entry`
2. Напишите манифест для создания **replicaSet** ui из ранее собранного Docker образа. После применения манифеста в **namespace** reddit-entry должны быть развернуты две реплики **pod** ui
3. Удалите один из **pod**‘ов, который был развернут с использованием манифеста (`kubectl delete pod ui-* -n reddit-entry`). Что произошло после удаления **pod**?

Практика

4. Отмасштабируйте количество реплик ui до трех с использованием ad-hoc команды (`kubectl scale replicaset ui --replicas=3 -n reddit-entry`), проверьте, что количество реплик увеличилось
5. Примените написанный ранее манифест заново. Почему подход с ad-hoc масштабированием не всегда оправдан?

Практика

Для написания манифеста необходимо указать версию api, в качестве kind установить ReplicaSet и указать название

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: ui

...
```

Указать количество реплик и поды

```
...

spec:
  replicas: 2
  selector:
    matchLabels:
      app: ui
```

Практика

И описать этот под

```
...
  template:
    metadata:
      name: ui
      labels:
        app: ui
    spec:
      containers:
      - name: ui
        image: <container name>
```

Задание

- Добавьте **requests** и **limits** для контейнера в манифест
- Добавьте задание произвольных ENV переменных в контейнере в манифест
- Убедитесь что ENV переменные внутри контейнера выставлены в соответствии с описанием в манифесте (предложите, как это можно сделать)

Практика

Установить **requests** в значение `memory: "256Mi"` и `cpu: "100m"`. Для этого, в манифест файл под `image` добавим:

```
...
  resources:
    requests:
      memory: "256Mi"
      cpu: "100m"
```

И добавим **limits** в `memory: "512Mi"` и `cpu: "200"`. Для этого под **requests** добавим:

```
...
    limits:
      memory: "512Mi"
      cpu: "200m"
```

Практика

В итоге получим:

```
...
image: <container name>
resources:
  requests:
    memory: "256Mi"
    cpu: "100m"
  limits:
    memory: "512Mi"
    cpu: "200m"
```

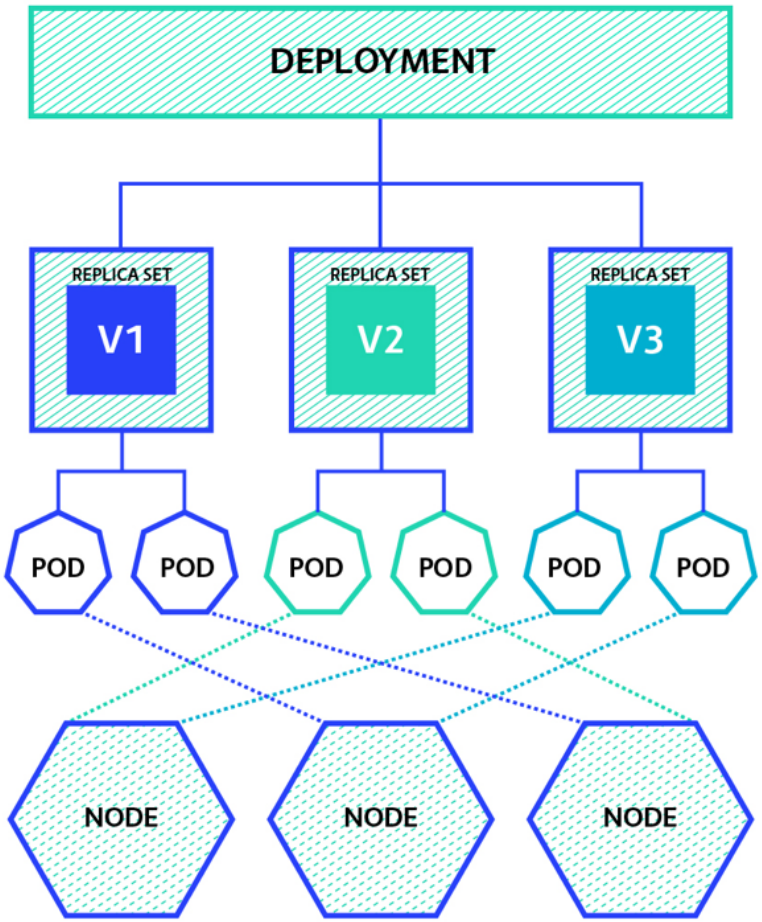
Практика

Добавим установку переменной `VERSION` в манифест файл. Для этого под **resources** пишем:

```
1   ...
2   env:
3   - name: VERSION
4     value: 1.0.0
5
```

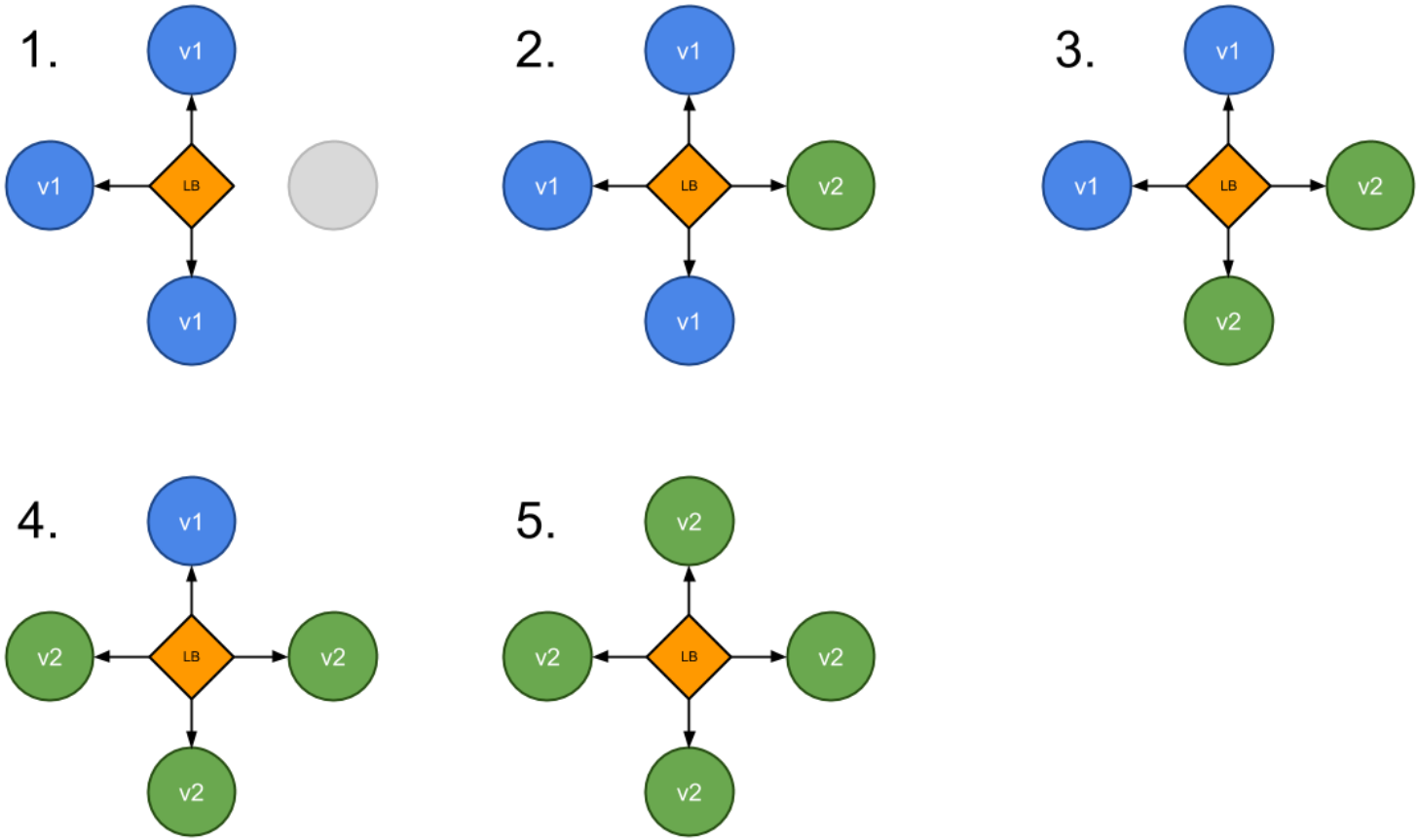
Deployments

Версионирование replicaSets



Deployments

Rolling updates



Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.8
```

Практика

1. Удалите созданный ранее **replicaSet**

2. Напишите манифест для создания **deployment** ui из ранее собранного Docker образа. После применения манифеста в **namespace** reddit-entry должны быть развернуты две реплики **pod** ui

> Манифест deployment почти не отличается от манифеста replicaSet

Практика

3. Добавьте новую environment переменную в спецификацию контейнера в манифесте **deployment**

4. Примените новую версию манифеста, отследите последовательность пересоздания **pod**

5. Проверьте количество **replicaSet** в namespace **reddit-entry**. Почему появилось 2 **replicaSet**? Проверьте, что внутри созданного контейнера действительно появилась указанная в манифесте environment переменная

> Для наблюдения в live режиме за объектами кластера целесообразно использовать ключ -w. В данной практике предполагается использование команды вида `kubectl apply -f deployment.yml && kubectl get pods -w`

Практика

Для написания deployment манифеста необходимо лишь изменить kind в манифесте replicaSet

Было:

```
1   apiVersion: apps/v1
2   kind: ReplicaSet
3   metadata:
4     name: ui
5   ...
```

Стало:

```
1   apiVersion: apps/v1
2   kind: Deployment
3   metadata:
4     name: ui
5   ...
```

Практика

Чтобы добавить environment, добавьте env в описание контейнера

```
...

spec:
  containers:
  - name: ui
    image: <container name>
    env:
    - name: uiVersion
      value: "1.0"
```

Практика

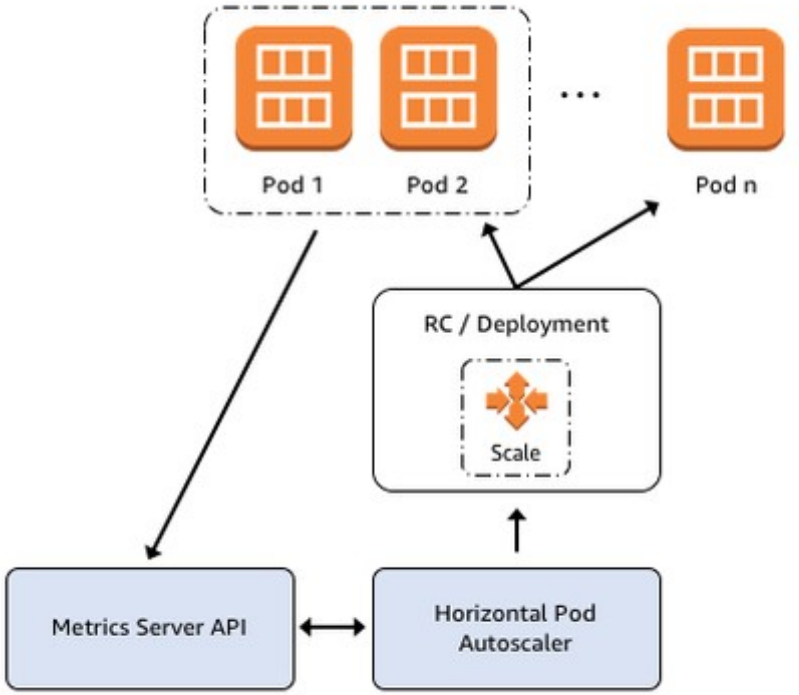
Rollback

Представьте, что по каким-то причинам понадобилось вернуться на старую версию манифеста

- Увеличьте версию до 2 и примените изменения
- Введите команду `kubectl rollout undo deployment ui -n reddit-entry` (откат на предыдущую версию **deployment**)
- Проверьте наличие environment переменной внутри контейнера
- Почему применение данного способа отката не всегда целесообразно?

Horizontal Pod Autoscalers (HPA)

- Автомасштабирование количества pod в deployment/replicaSet в зависимости от нагрузки



Практика

Создайте манифест со следующим содержимым и примените его:

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: ui
  namespace: reddit-entry
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: ui
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: memory
      targetAverageUtilization: 1
```

Практика

- Отследите, какой процент RAM от requests в действительности потребляют реплики ui (`kubectl get hpa -n reddit-entry/kubectl describe hpa ui -n reddit-entry`)
- Измените манифест таким образом, чтобы количество реплик уменьшилось до одной, и примените его (придется подождать для процедуры scale-down)

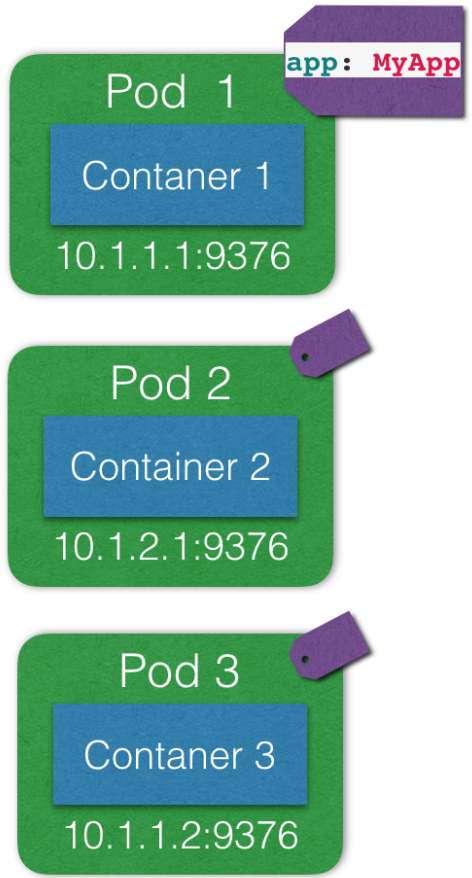
Service

Поды:

- Смертны
- Динамически создаются и удаляются
- IP-адреса появляются и исчезают вместе с Pod’ами
- Несколько Pod’ов могут выполнять одну и ту же функцию

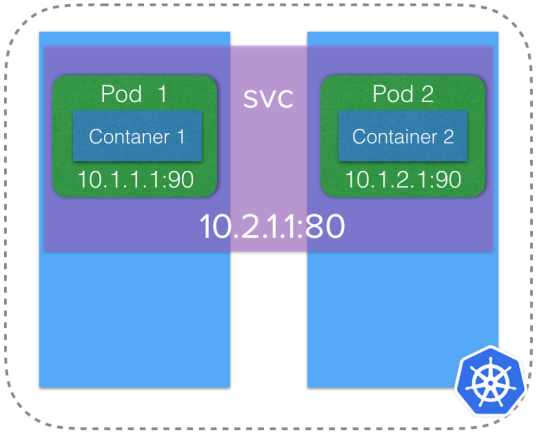
Service

- Абстракция, описывающая набор Pod’ов и конфигурацию доступа к ним
- Позволяет отвязаться от использования конкретных Pod’ов



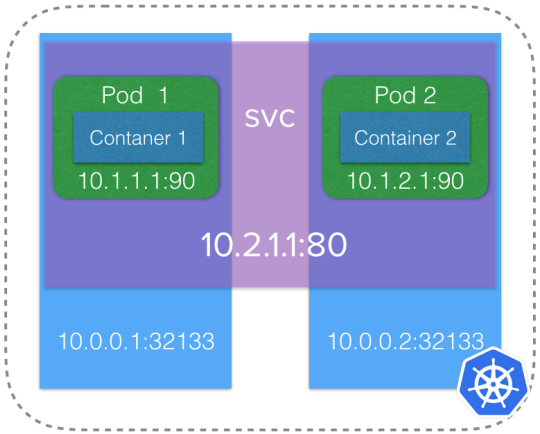
Service | ClusterIP

```yaml
kind: Service
apiVersion: v1
metadata:
  name: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```



Service | NodePort

```yaml
kind: Service
apiVersion: v1
metadata:
  name: nginx-node
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```



DNS

- Обращение к сервису внутри namespace:

  `<service_name>`

- Обращение к сервису внутри кластера:

  `<service_name>.<namespace>`

- FQDN сервиса:

  `<service_name>.<namespace>`.svc.cluster.local

- FQDN пода:

  `<pod_ip>.<namespace>`.pod.cluster.local

Практика

- Напишите манифест для создания **service** ui типа **NodePort**. После применения манифеста в **namespace** reddit-entry трафик при обращении к сервису по имени изнутри кластера должен балансироваться между подами ui. Компонент ui слушает TCP/9292
- Узнайте адрес одного из хостов вашего кластера (`kubectl get nodes -o wide`)
- Узнайте порт хоста, на который пробросился порт контейнера (`kubectl get svc -n reddit-entry`)
- Попробуйте получить доступ через web к интерфейсу ui

Практика

По аналогии с другими манифестами необходимо указать версию api, kind и название

```
1   apiVersion: v1
2   kind: Service
3   metadata:
4     name: ui
5   ...
```

Затем необходимо указать в качестве типа Service NodePort, и то что данный сервис применяется к ui

```
1   ...
2   spec:
3     type: NodePort
4     selector:
5       app: ui
6   ...
```

Практика

И останется лишь указать порт внутри docker контейнера и порт для взаимодействия внутри кластера k8s

```
1   ...
2     ports:
3     - protocol: TCP
4       port: 9292
5       targetPort: 9292
```

Ingress

Объект управляющий внешним доступом к сервисам внутри кластера. Обеспечивает:

- Организацию единой точки входа в приложения снаружи
- Балансировку трафика
- Терминацию SSL
- Виртуальный хостинг на основе имен и т.д.

Работает на L7 уровне.

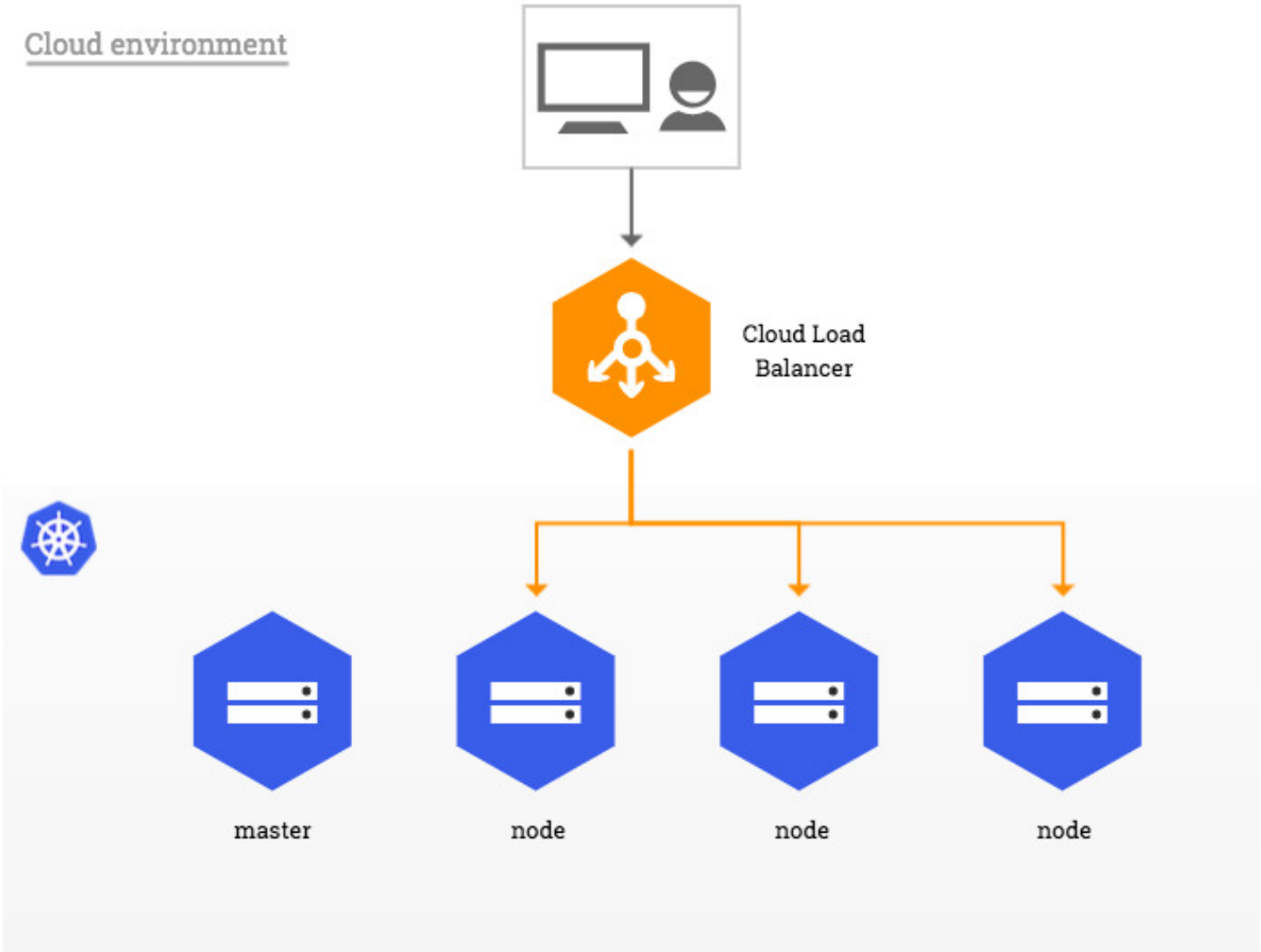
Ingress

Ingress - набор правил внутри кластера Kubernetes.

Для применения данных правил нужен **Ingress Controller** - плагин который состоит из 2-х функциональных частей:

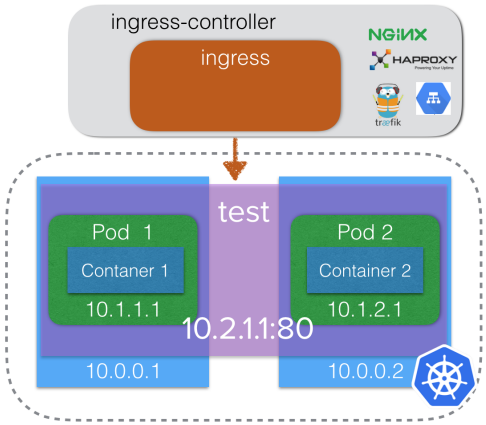
- Приложение, которое отслеживает через Kubernetes API новые объекты Ingress и обновляет конфигурацию балансировщика
- Балансировщик (nginx, HAProxy, traefik,...), который отвечает за управление сетевым трафиком

Ingress


Cloud environment
Cloud Load Balancer
master
node
node
node

Ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
  labels:
    app: test
spec:
  rules:
  - http:
      paths:
      - path: /*
        backend:
          serviceName: test
          servicePort: 80
```



Практика

- Напишите манифест для создания **ingress** ui. После применения манифеста в **namespace** reddit-entry вывод команды `kubectl get ingress -n reddit-entry` должен выглядеть следующим образом:

```
NAME      HOSTS     ADDRESS          PORTS     AGE
ui        *         <IP_ADDRESS>     80        20m
```

- Зайдите на присвоенный IP-адрес и убедитесь, что web интерфейс ui доступен

> LB в GCP работает с некоторой задержкой

Практика

По аналогии с другими манифестами необходимо указать версию api, kind и название

```
1   apiVersion: extensions/v1beta1
2   kind: Ingress
3   metadata:
4     name: ui
5     labels:
6       app: ui
```

Практика

Теперь необходимо описать правило для ingress. Укажем, что у нас есть правило для http трафика. Оно подпадает под все маршруты, работает для сервиса с именем ui и открывает 80 порт.

```
spec:
  rules:
  - http:
      paths:
      - path: /*
        backend:
          serviceName: ui
          servicePort: 80
```

Чистка за собой

- Текущие объекты Kubernetes нам дальше не понадобятся. Манифесты сохраните.
- Если у вас получилось зайти в ваш Reddit сервис, то удалите созданные вами сущности в Kubernetes
- Если вы вели работу в Namespace `reddit-entry`, то самый простой способ очистки - удалить Namespace

```
1  kubectl delete ns reddit-entry
```

Volumes

- Меньше ограничений по сравнению с Docker’ом
- Pod может использовать несколько типов Volume’ов одновременно

Local Volumes

- **emptyDir** - пустая директория, создается на Node и удаляется вместе с удалением Pod. Используется для обмена данными между контейнерами внутри Pod
- **hostPath** - директория на Node, не удаляется при удалении Pod. Используется для доступа к информации на Node, например к /var/lib/docker/
- **local** - директория на Node, но есть отличия от hostPath

emptyDir

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: gcr.io/google_containers/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

ConfigMap

Используется для монтирования конфигурации внутрь контейнера

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  custom.conf: |
    <place you config here>
```

ConfigMap Mount

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.15.8
      volumeMounts:
      - name: config
        mountPath: /etc/nginx/conf.d/custom.conf
  volumes:
    - name: config
      configMap:
        name: nginx-config
```

Volume Types

Cloud	Custom
AWS EBS	NFS
AzureDisk и AzureFile	CephFS и GlusterFS
gcePersistentDisk	FC и iSCSI

gcePersistentDisk

- Надо предварительно создать диск в GCP*
- После удаления pod данные на диске останутся

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: nginx
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    gcePersistentDisk:
      pdName: my-data-disk
      fsType: ext4
```

Persistent Volumes

- PersistentVolume
- PersistentVolumeClaim
- StorageClass

Persistent Volumes

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

PersistentVolumeClaim

- Запрос на хранилище от пользователя
- PVC могут требовать определенный объем хранилища и прав доступа
- Создаются на уровне namespace

PersistentVolumeClaim

- Access Modes
- Resources
- Selector
- Class

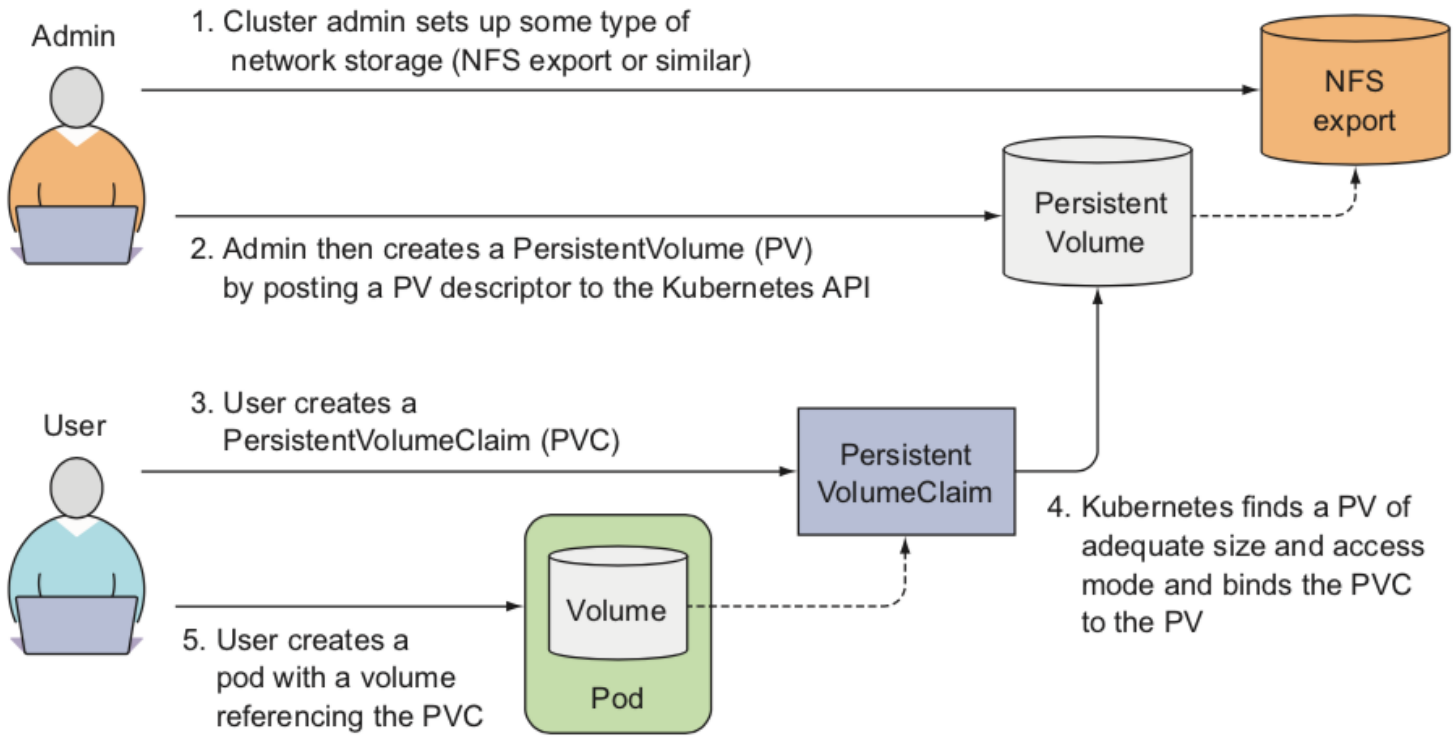
PersistentVolumeClaim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
  - ReadWriteOnce
  selector:
    matchLabels:
      release: "stable"
```

PVC as Volume

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim:
      claimName: mongodb-pvc
```

Static Workflow



Storage Classes

- Описание “классов” различных систем хранения
- Разные классы могут использоваться для:
  - Произвольных политик
  - Динамического provisioning

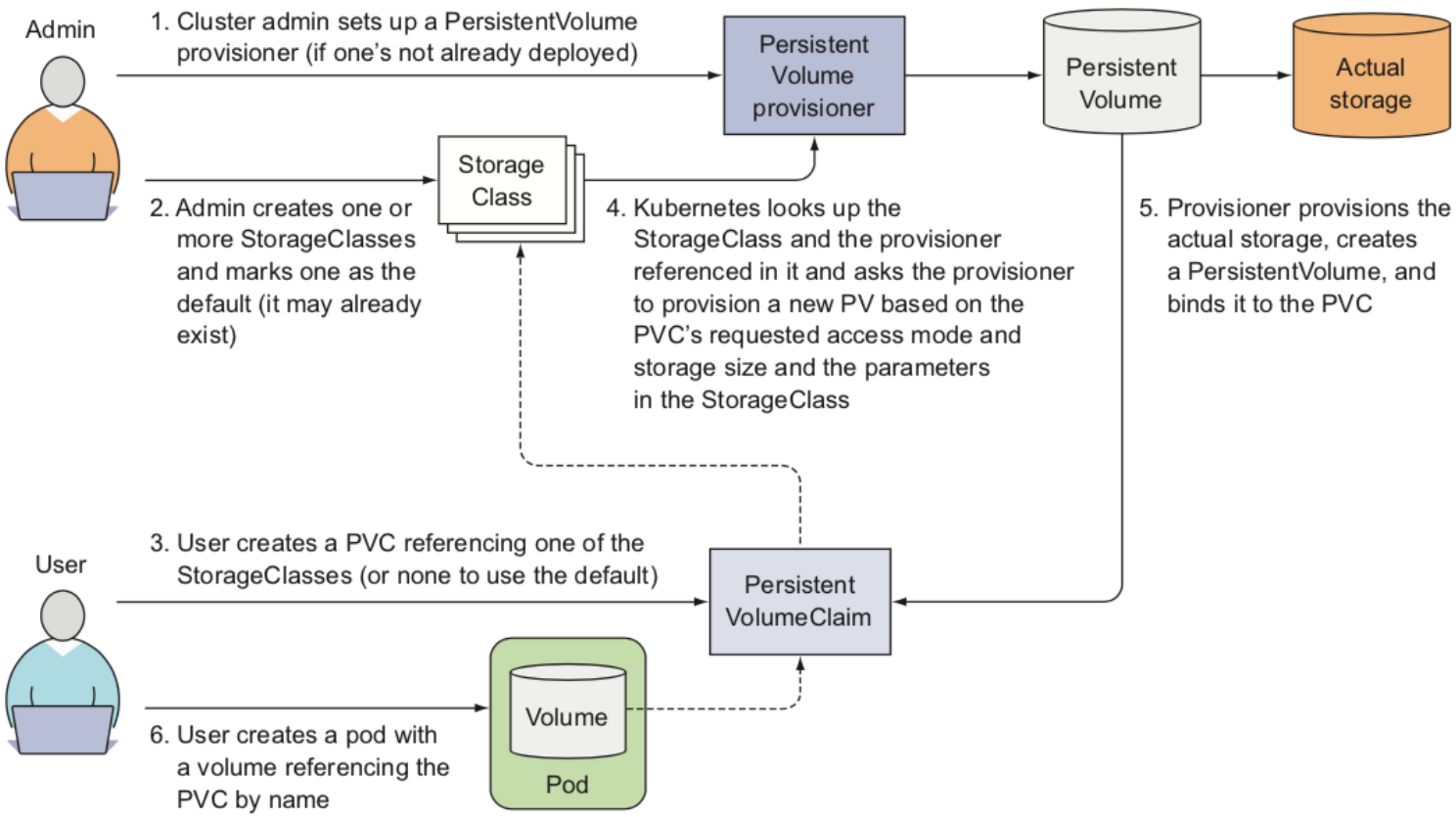
Storage Classes

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  zone: europe-west1-c
```

PersistentVolumeClaim с StorageClass

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  storageClassName: fast
  resources:
    requests:
      storage: 100Mi
  accessModes:
    - ReadWriteOnce
```

Dynamic Workflow



Dynamic Volume Provisioning

- Использует StorageClass для provisioning’а
- PersistentVolumeClaim содержит информацию о нужном StorageClass

Некоторые практики разработки приложений

- Сервисы не должны запускаться от `USER=root`. Учитывайте данный факт при написании приложений
- Каждый компонент продукта должен корректно обрабатывать сигналы операционной системы (SIGTERM)
- Каждый процесс приложения должен быть запущен в отдельном контейнере, связанные контейнеры должны быть объединены в один POD
- Приложения должны писать логи в stdout/stderr (подробнее - в секции про мониторинг)