Volumes, Storages, Stateful приложения

Не забудь включить запись!

Зачем нужны Volumes?

- Поды эфемерны
- Некоторые данные необходимо хранить
- Контейнерам внутри пода необходимо обращаться к одним и тем же файлам

Что такое Volume?

Volume - абстракция реального хранилища

- Volume создается и удаляется вместе с подом
- Один и тот же Volume может использоваться одновременно несколькими контейнерами в поде

Типы Volume | emptyDir

- Изначально пустой каталог на хосте
- Используется для обмена файлами между разными контейнерами одного pod
- Данные могут храниться в tmpfs

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web
spec:
  initContainers:
    - name: init
      image: busybox:1.31.0
      command: ['...']
      volumeMounts:
        - mountPath: /app
          name: app
  containers:
    - name: web
      image: web:1.0.1
      volumeMounts:
        - mountPath: /app
          name: app
  volumes:
    - name: app
      emptyDir: {}
```

Типы Volume | hostPath

- Возможность монтировать файл или директорию с хоста
- Часто используется для служебных сервисов (Node Exporter, Fluentd/Fluent Bit, etc...)
- Scheduler не учитывает hostPath в своих алгоритмах

```
1  apiVersion: apps/v1
2  kind: DaemonSet
3  metadata:
4    name: fluentd
5  spec:
6    ...
7    template:
8      spec:
9        containers:
10       - name: fluentd
11         image: fluent/fluentd-kubernetes-daemonset
12         volumeMounts:
13         - name: varlog
14           mountPath: /var/log
15       volumes:
16       - name: varlog
17         hostPath:
18           path: /var/log
```

Облачные Volumes

- AWS EBS
- AzureDisk и AzureFile
- gcePersistentDisk

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
```

Прочие Volumes

- nfs
- CephFS и GlusterFS
- FC и iSCSI
- ...

Прочие Volumes

- ConfigMap - хранят:
  - конфигурацию приложений
  - значения переменных окружения отдельно от конфигурации пода
- Secret - хранят чувствительные данные (возможно шифрование содержимого в etcd, но в манифестах - base64)

Оба типа функционируют схожим образом:

- Сначала создаем соответствующий ресурс (ConfigMap, Secret)
- В конфигурации пода в описании volumes или переменных окружения ссылаемся на созданный ресурс

Volume plugins

- In-tree - плагины вкомпилированы в Kubernetes
- FlexVolume - первая попытка создания out-of-tree exec based плагина для расширения возможностей Kubernetes
- CSI (Container Storage Interface) - стандартизованный интерфейс для реализации пользовательских плагинов

> В настоящее время идет миграция некоторых in-tree плагинов на CSI. Подробности в блоге Kubernetes

Persistent Volumes

- Создаются на уровне кластера
- PV похожи на обычные Volume, но имеют отдельный от сервисов жизненный цикл

PersistentVolume

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Описание

PersistentVolumeClaim

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

PersistentVolumeClaim

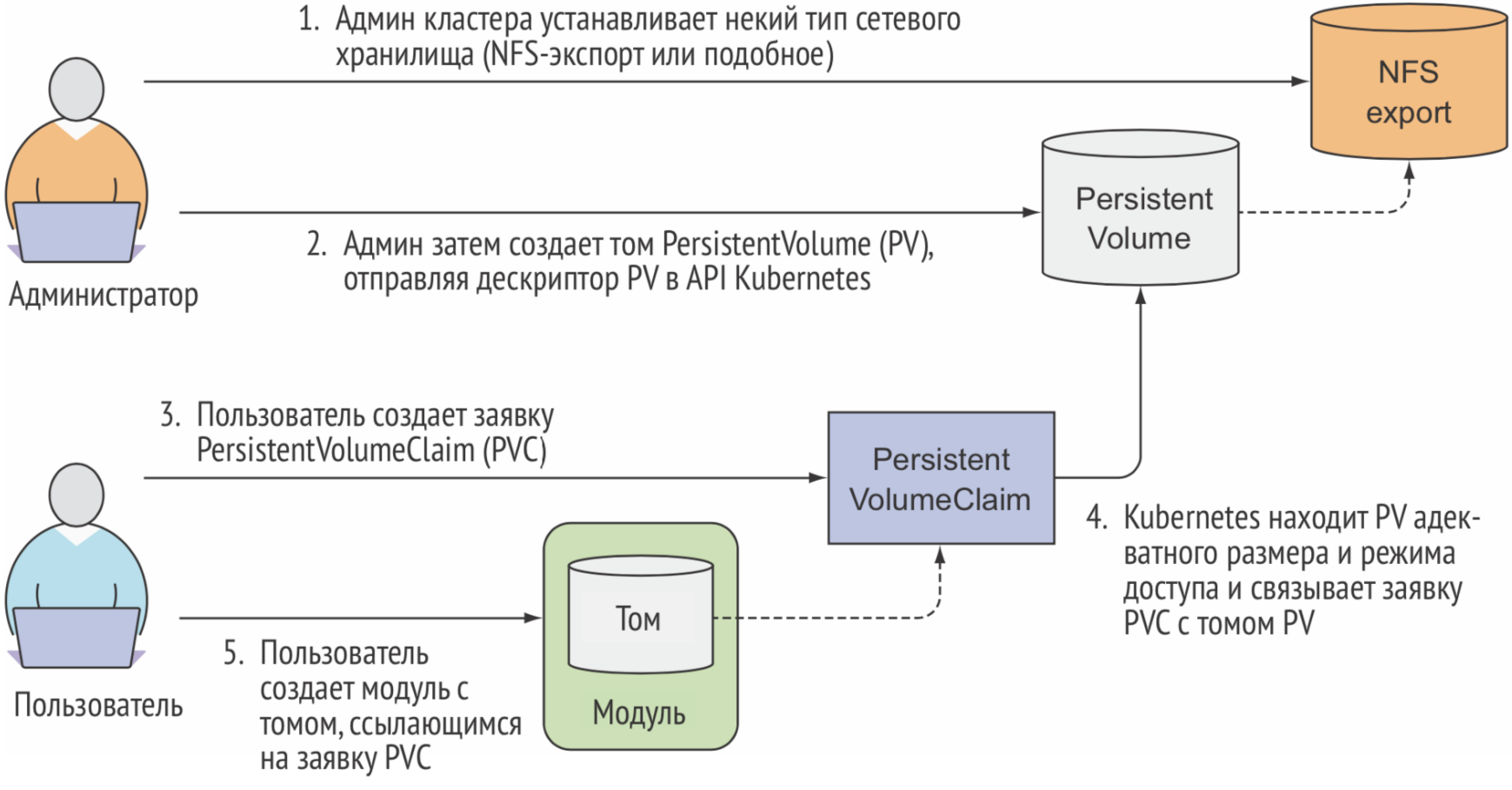
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
```

Описание

Claims as Volumes

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: mypod
5  spec:
6    containers:
7      - name: myfrontend
8        image: nginx
9        volumeMounts:
10       - mountPath: "/var/www/html"
11         name: mypd
12   volumes:
13     - name: mypd
14       persistentVolumeClaim:
15         claimName: myclaim
```

Схема создания PV и PVC



В какой момент происходит монтирование

- Kubernetes монтирует сетевой диск на ноду
- Runtime пробрасывает том в контейнер

Local Persistent Volume

- Некоторый аналог, hostPath, но есть отличия:
  - Scheduler учитывает наличие local volumes в своих алгоритмах
  - Ссылка на local volume из манифеста pod возможна только через PVC
  - Возможно монтирование raw block devices

Подробности

Storage Classes

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

StorageClass

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
```

External Storage

Политики переиспользования PV

PV может иметь несколько разных политик переиспользования ресурсов хранилища:

- **Retain** - после удаления PVC, PV переходит в состояние “released”, чтобы переиспользовать ресурс, администратор должен вручную удалить PV, освободить место во внешнем хранилище (удалить данные или сделать их резервную копию)
- **Delete** - (плагин должен поддерживать эту политику) PV удаляется вместе с PVC и высвобождается ресурс во внешнем хранилище
- **Recycle** - удаляет все содержимое PV и делает его доступным для использования (**deprecated**)

Изменение размера PV

```
 1 apiVersion: v1
 2 kind: PersistentVolumeClaim
 3 metadata:
 4   name: myclaim
 5 spec:
 6   accessModes:
 7     - ReadWriteOnce
 8   volumeMode: Filesystem
 9   resources:
10     requests:
11 -       storage: 8Gi
12 +       storage: 10Gi
13   storageClassName: slow
```

Режимы доступа (Access Modes)

Тома монтируются к кластеру с помощью различных провайдеров, они имеют различные разрешения доступа чтения/записи, PV дает общие для всех провайдеров режимы.

PV монтируется на хост с одним их трех режимов доступа:

- ReadWriteOnce - **RWO** - только один узел может монтировать том для чтения и записи
- ReadOnlyMany - **ROX** - несколько узлов могут монтировать том для чтения
- ReadWriteMany - **RWX** - несколько узлов могут монтировать том для чтения и записи

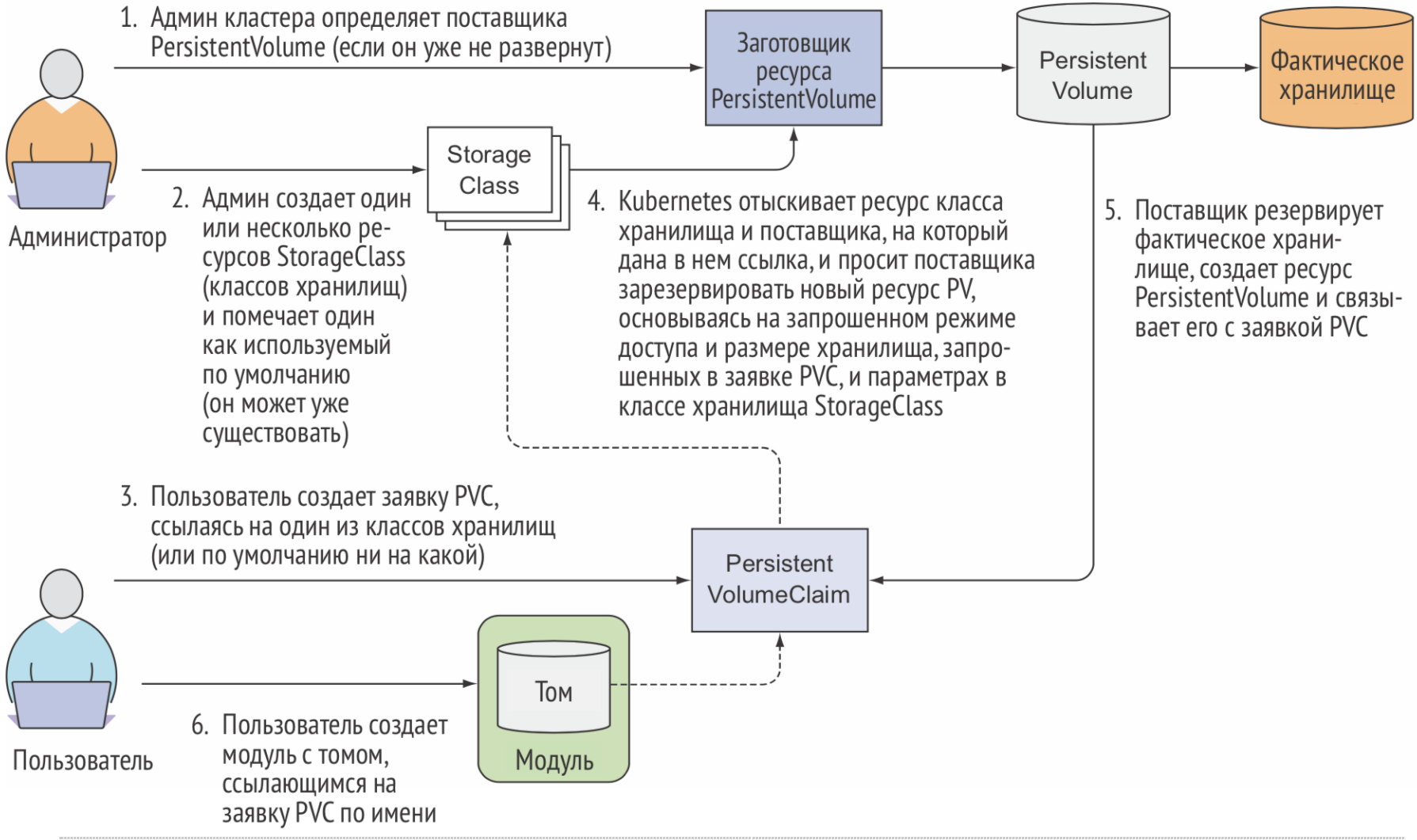
Лимитирование ресурсов

Администратор кластера может ограничить:

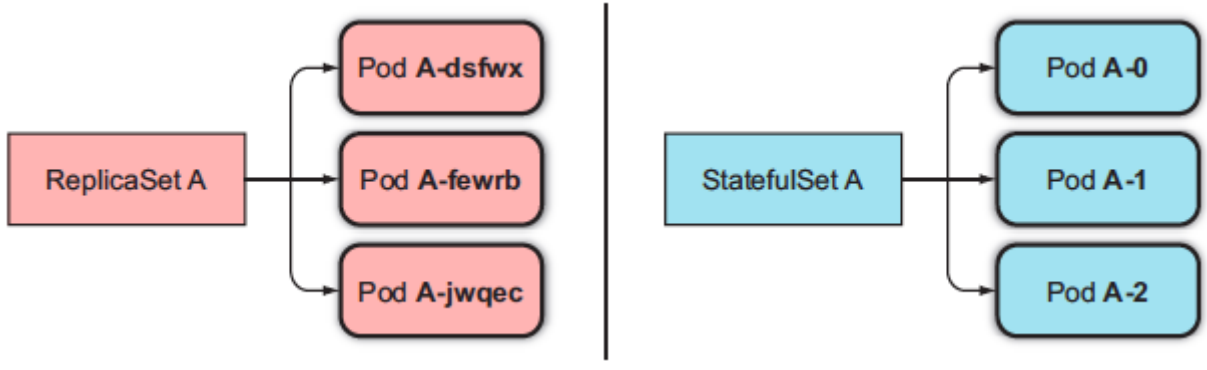
- Количество PVC в неймспейсе
- Размер хранилища, который может запросить PVC
- Объем хранилища, который может иметь неймспейс

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: storagelimits
spec:
  limits:
  - type: PersistentVolumeClaim
    max:
      storage: 2Gi
    min:
      storage: 1Gi
```

Схема создания PV и PVC



StatefulSet



Поды в StatefulSet относятся к "питомацам", а не "стаду", поэтому:

- Каждый под имеет уникальное состояние (имя, сетевой адрес, volumes)
- Для каждого создается отдельный PVC

Конфигурация StatefulSet

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
spec:
  selector:
    matchLabels:
      app: mongodb
  serviceName: "mongodb"
  replicas: 3
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: mongodb
        image: mongo:3.4.21-xenial
        ports:
        - containerPort: 27017
          name: db
        volumeMounts:
        - name: dbstorage
          mountPath: /data/db
...
```

Конфигурация StatefulSet (продолжение)

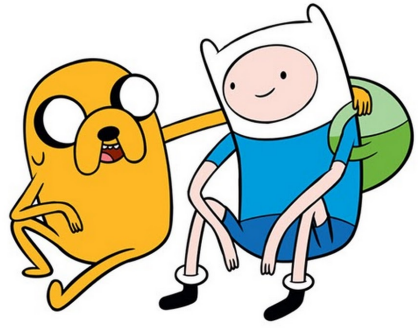
```
volumeClaimTemplates:
  - metadata:
      name: dbstorage
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Особенности StatefulSet

- Так как поды в StatefulSet имеют разное состояние для обеспечения сетевой связности должен использоваться Headless Service
- Тома для подов должны создаваться через PersistentVolume
- Удаление/масштабирование подов не удаляет тома, связанные с ними

Особенности StatefulSet (продолжение)

- Масштабирование выполняется постепенно, следующий под будет создан только вслед за предыдущим
- У каждого pod свой pvc и свой pv, поэтому надо пользоваться volume claim template
- Если pod оказался на авайриной ноде - поведение будет отличаться от поведения в deployment
- Уникальные, предсказуемые имена pod



Спасибо за внимание!

Время для ваших вопросов!