Volumes, Storages, Stateful-приложения

Не забудь включить запись!

Зачем тома?

- Поды эфимерны
- Нужно хранить данные
- Контейнерам внутри пода необходимо обращаться к одним и тем же файлам

Что такое том?

Том - абстракция реального хранилища

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

Типы томов

- emptyDir – просто пустой каталог
- hostPath – том, который связывается с каталогом на host узле
- local - том, связанный с разделом/диском на host узле, и привязывающий под к этому конкретному узлу (1.14 stable), дает более высокую производительность, чем remote storages
- gcePersistentDisk - диск из GCP
- fc - fibre chanel
- PersistentVolumeClaim - запрос на PV
- CSI - Container Storage Interface
- flexVolume - интерфейс для плагинов storages, появился в версии 1.2 до CSI

Kubernetes Volumes

"Служебные" типы

- ConfigMap - словари конфиуграции, хранят:
  - конфигурацию приложений
  - значения переменных окружения отдельно от конфигурации пода
- Secret - хранят чувствительные данные (данные в etcd зашифрованы с версии 1.7)

Оба типа функционируют схожим образом:

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

Определение тома в конфигурации пода

```
 1 apiVersion: v1
 2 kind: Pod
 3 metadata:
 4   name: test-pd
 5 spec:
 6   containers:
 7   - image: k8s.gcr.io/test-webserver
 8     name: test-container
 9     volumeMounts:
10     - mountPath: /test-pd
11       name: test-volume
12   volumes:
13   - name: test-volume
14     # This GCE PD must already exist.
15     gcePersistentDisk:
16       pdName: my-data-disk
17       fsType: ext4
```

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

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

Описание

Claims as Volumes

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

В какой момент просиходит монитирование

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

StorageClass

```
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 высвобаждается ресурс во внешнем хранилище
- Recycle - удаляет все содержимое PV и делает его доступным для использования (**depricated**)

Изменение размера PV

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
-      storage: 8Gi
+      storage: 10Gi
  storageClassName: slow
```

Создание PVC без указания класса хранилища

В заявке PVC можно не указывать конкретное имя storage class, в таком случае будет использоваться storage class выбранный по умолчанию, чтобы не создавать новый PV, а использовать существующий надо указать в PVC storageClassName: “”

Volume Cloning

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-clone
  Namespace: demo-namespace
spec:
  storageClassName: csi-storageclass
  dataSource: #Источник для клонирования
    name: src-pvc
    kind: PersistentVolumeClaim
    apiGroup: ""
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

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

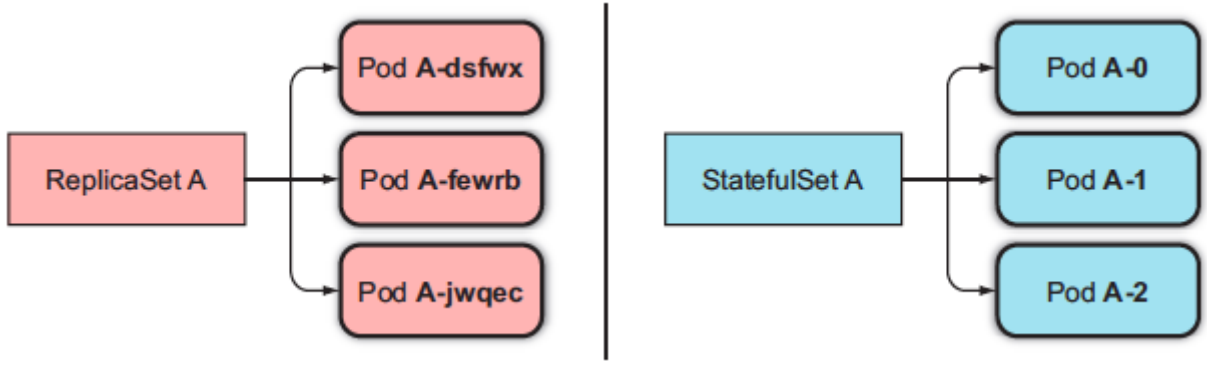
Лимитирование ресурсов

Администратор кластера может ограничить:

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

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

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

```
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

- Т.к. 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