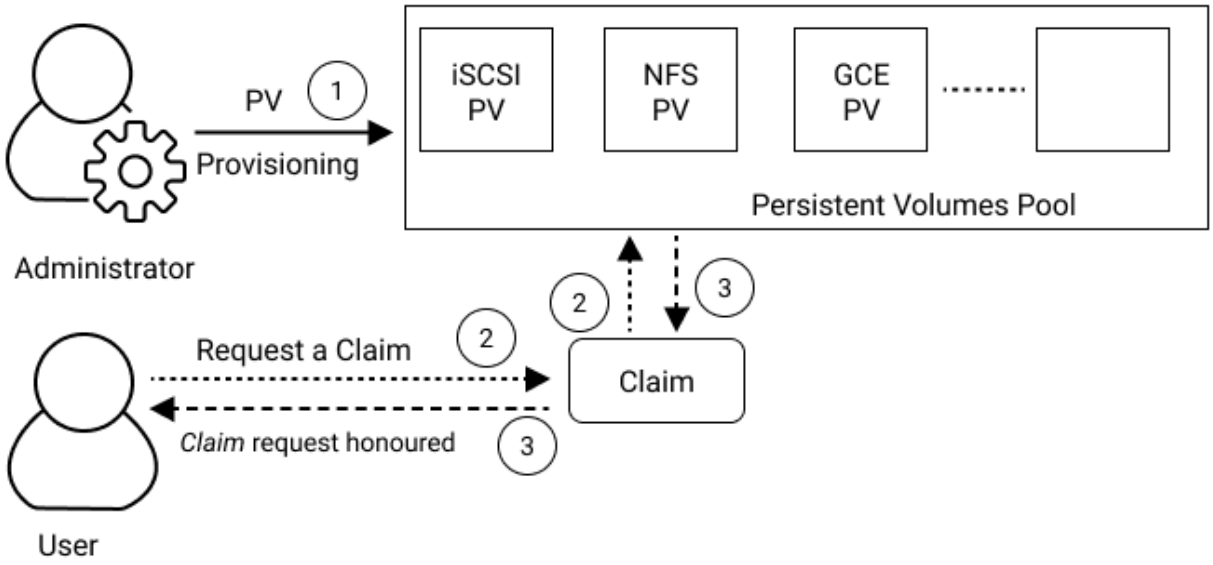
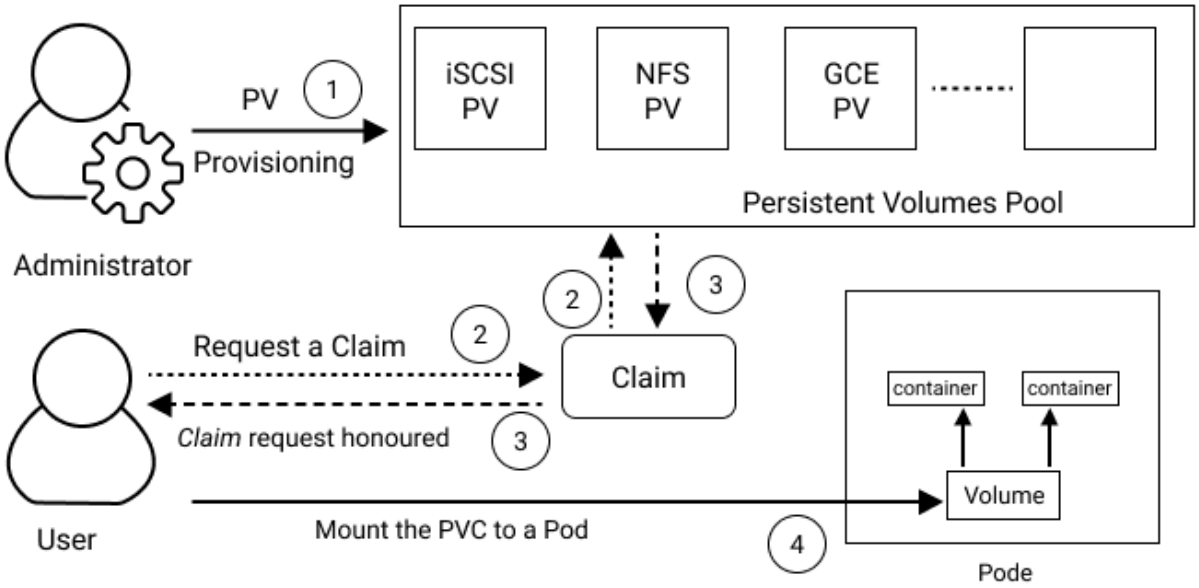
K8S 29: Kubernetes. Resources. PersistentVolume (Claims)

Описание:

В предыдущем задании мы с вами не разобрали еще один вид Volume, который предоставляет возможности очень гибкой работы с хранилищем под требуемые нужды, но требует более сложной настройки.
Обычно администратор хранилища выделяет пространство пользователям. А затем пользователи используют выделенное хранилище для своих нужд. В Kubernetes для этого служит PersistanceVolume. Данная абстракция предоставляет API для управления системами хранения (NFS, GlusterFS, CephFS, iSCSI — вот лишь небольшой и ненеполный список). В отличие от Docker, плагины для работы с разными системами хранения уже включены в код Kubernetes. Ничего дополнительного устанавливать не нужно (но расширять список можно при помощи CSI - Container Storage Interfaces — плагинов, которые предоставляют общий стандартный интерфейс для запросов к системам хранилища со стороны Kubernetes).
Каждый pod может запросить требуемое пространство из PV, используя Persistent Volume Claim (сокращенно — PVC).



При успешном запросе будет выделено требуемое пространство.



Для чего такие сложности? Кластер может использовать большое количество разных PV. Например, в облаке AWS есть десятки классов для дисков, все они дают разную производительность и стоят соответственно. Для логов можно использовать недорогие шпиндельные диски, для cassanda — взять ssd. Kubernetes будет автоматически управлять всеми дисками, снимая с вас работу по созданию дисков.
Ручное создание PV на примере локальных хранилищ. Создадим PV:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Данное хранилище будет использовать папку /mnt/data.
Теперь создадим PVC:

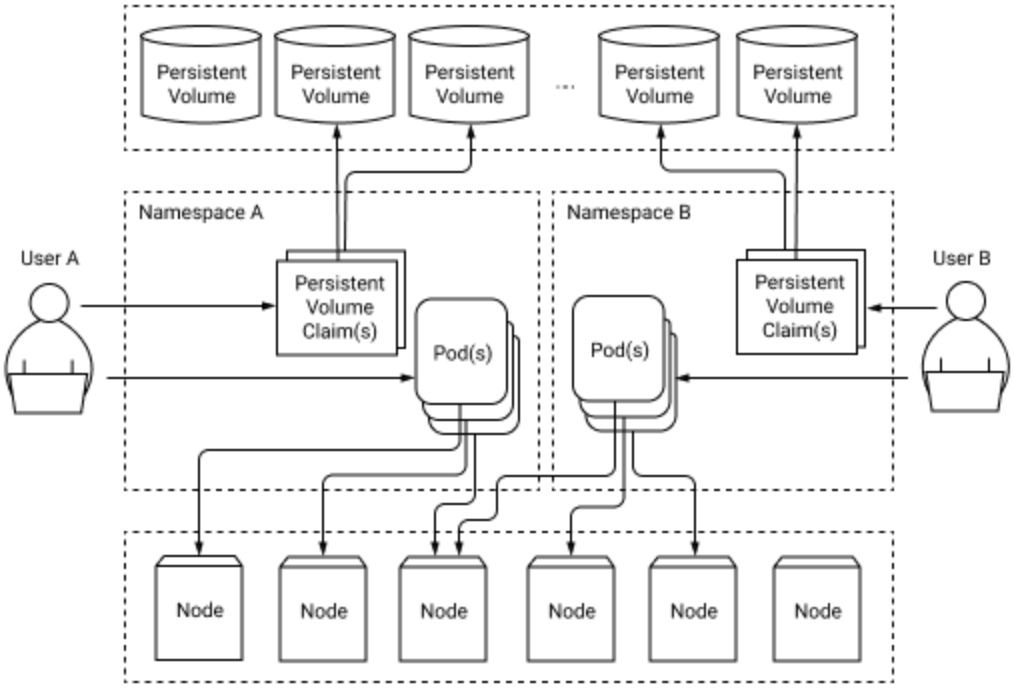
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

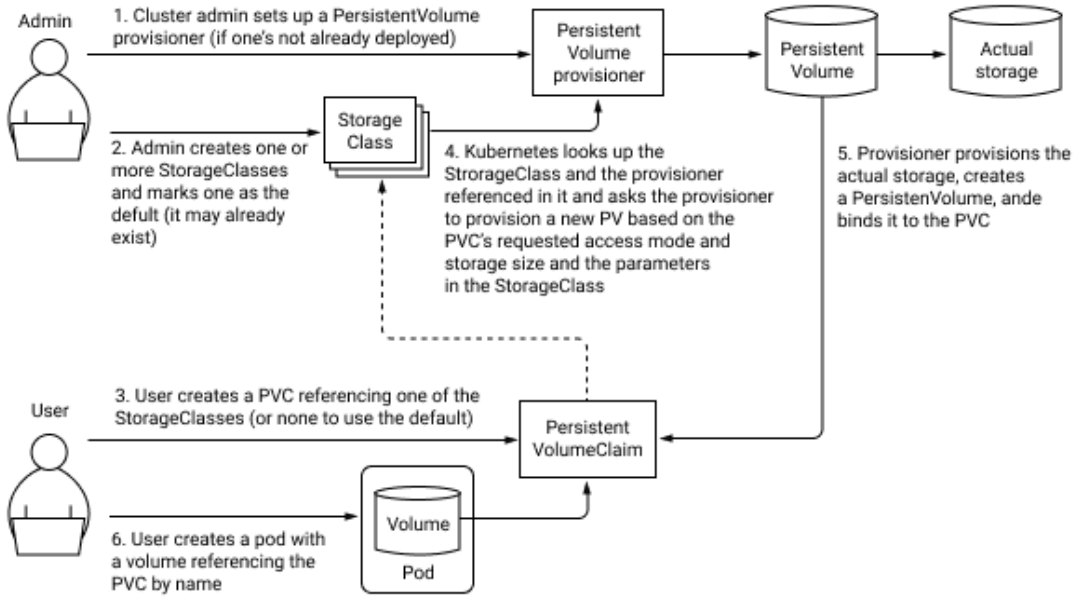
Мы запросили 3 Гб из storageclass=manual, данный storageclass обслуживается только одним хранилищем (создали ранее). Из хранилища в 10 Гб будет запрошено 3 Гб. А теперь подключим хранилище к приложению:

```
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Теперь при запуске контейнера nginx папка /usr/share/nginx/html будет примонтирована на данный PV. Почти как docker bind mount. При добавлении файлов в эту папку их можно получить через nginx.



Мы почти все делали в ручном режиме. Для автоматического создания PV есть определенные типы Provisioner. Например, при запуске в облаке AWS можно автоматически создавать PV. Это делается через абстракцию StorageClass.



StorageClass — это набор параметров для конкретного Provisioner. Provisioner — это программа, которая взаимодействует с инфраструктурой, создает диски и делает PV. StorageClass автоматически делает PVC.
А теперь попробуем автоматический Provisioner:

```
kubectl apply -f
https://raw.githubusercontent.com/rancher/local-path-provisioner/maste
r/deploy/local-path-storage.yaml
```

Будет создан storageclass local-path и provisioner для него в namespace local-path-provisioner. Точно также можем в ручном режиме создать PVC и pod:

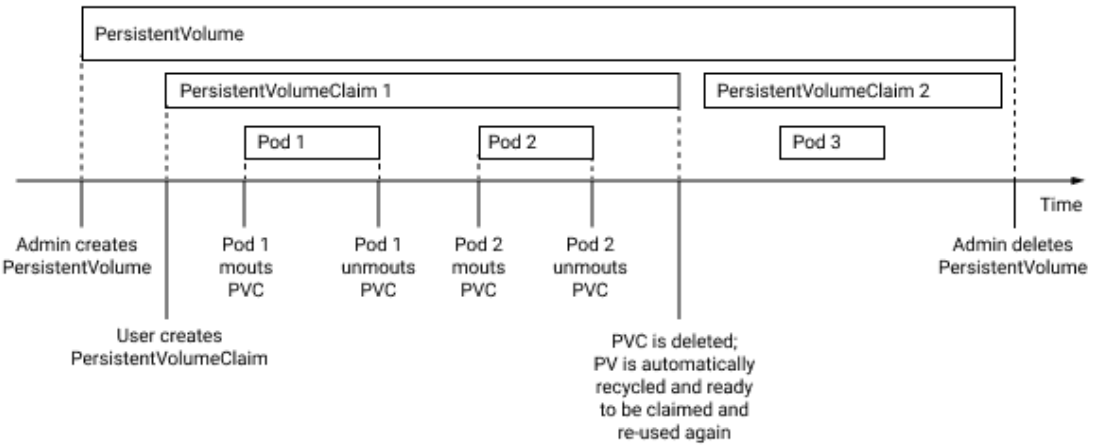
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-path-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-path
  resources:
    requests:
      storage: 2Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
  namespace: default
spec:
  containers:
  - name: volume-test
    image: nginx:stable-alpine
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - name: volv
      mountPath: /data
    ports:
    - containerPort: 80
  volumes:
  - name: volv
    persistentVolumeClaim:
      claimName: local-path-pvc
```

А теперь сделаем все автоматически в рамках deployment:

```yaml
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  serviceName: nginx-server
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        image: "nginx:alpine-stable"
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - containerPort: 80
          name: http
        readinessProbe:
          httpGet:
            path: /healthz
            port: 80
          initialDelaySeconds: 10
          timeoutSeconds: 2
          periodSeconds: 5
        livenessProbe:
          httpGet:
            path: /healthz
            port: 80
          initialDelaySeconds: 120
          timeoutSeconds: 2
          periodSeconds: 5
        volumeMounts:
        - mountPath: /data
          name: nginx-vol
      restartPolicy: Always
```

```
volumeClaimTemplates:
- metadata:
    name: nginx-vol
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi
    storageClassName: local-path
```

Самая важная часть — это spec.volumeClaimTemplates. Мы описали, что желаем получить 1 Гб диск класса local-path. При отправке данного yaml в API-server local-path-provisioner автоматически создаст PV, PVC, а kubelet подключит хранилище в pod.
Время жизни PV PVC



Хотелось бы еще упомянуть, как работает расширение хранилищ, - в последних версиях Kubernetes расширение PVC стало просто, как никогда. Требуется просто изменить размер PVC, после чего связанный с ним PV расширится (если это позволяет инфраструктура - можно отслеживать через describe pvc) и потребуется просто перезапустить под для применения нового объема. Но есть 2 НО - не все виды хранилищ поддерживают расширение (к примеру, GlusterFS позволяет, а NFS — нет), и остается необходимость для применения изменений в перезапуске пода.

Полезные ссылки:

- Persistent Volumes (official docs)
- Container Storage Interface (CSI) for Kubernetes GA (official blog)

Задание:

1. Напишите следующие манифесты:
- PV типа hostPath с именем rbmk8s28pv;
- связанный с ним PVC с именем rbmk8s28pvc;
- deployment с образом nginx:stable с подключением PVC как Volume в директорию /pvc с именем rbmk8s28deployment.
2. Примените данные манифесты в вашем Minikube-кластере и выведите их в кластере через describe (команды и вывод сохраните).
3. Создайте файл thisismyfile в директории, где подключен PVC изнутри контейнера, после этого выведите список файлов на хосте в директории, которая используется для PV (команды и вывод сохраните).
4. Перезапустите Pod, созданный для Deployment, и перепроверьте наличие файла в директории PVC (команды и вывод сохраните).
5. На проверку отправьте манифесты и сохраненные выводы и команды.