Надеюсь ты помнишь как мы рассматривали общие ресурсы в кластере - процессор и оперативку.

На самом деле этих ресурсов немножко больше.

В этой главе мы с тобой рассмотрим общие дисковые хранилища.

Когда ты работаешь с контейнерами на локальной машине, то вопрос дискового хранилища не встает. Оно же есть на сервере и мы его используем =).

Это локальное хранилище.

В k8s тоже можно делать локальные хранилища.

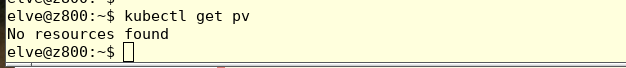
Но в кластере, когда узлы одного проекта могут быть размазаны по нескольким нодам, требуется также внешнее общее хранилище. И сегодня мы научимся с ним работать.

Для чего вообще эти все диски и хранилища нужны? Для сохранения данных при перезапуске контейнеров и для обмена даннными между контейнерами. Другими словами - для постоянного хранения (а не временного).

Потому сущность так и называется - **PersistentVolume**

Посмотреть список PersistentVolume можно с помощью команды

**kubectl get pv**



PersistentVolume, можно рассматривать как диск. По сути это и есть какое-либо дисковое хранилище. Либо единичное, либо распределенное.

Kubernetes позволяет подключать диски по следующим протоколам:

- NFS

- iSCSI

- DRBD

- Ceph

- GlusterFS

- local (локальное хранилище. любое. в т.ч. подключенное по FiberChannel).

Итак. У нас есть диск. Как его делить между контейнерами? Тут все делается по аналогии с обычной разметкой дисков на компьютере - делим диск на что-то вроде разделов. Эти разделы называются **PersistentVolume Claim.**

Если у тебя в манифесте для твоего проекта прописано, что требуется PVC, то при его запуске Kubernetes постарается выделить нужное место на одном из имеющися PV

Для создания и удаления PVC на PV используется такая сущность как Provisioner. Именно эта сущность занимается выделением пространства и его освобождением.

Для распределенных файловых систем, у которых есть API в k8s есть встроенные Provisioner-ы.

Поэтому при наличии таких PV дисковое пространство выделяется автоматически.

А вот для таких PV как NFS или Local нет provisioner-ов.

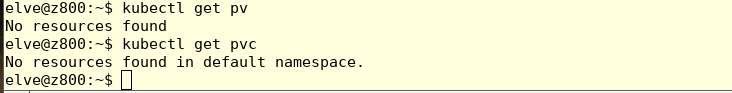
Поэтому если ты подключаешь диск NFS, то нужно будет самому порезать его на PVC.

А Kubernetes будет выдавать уже эти нарезанные PVC. И неважно, что там может быть избыточный объем. Отдается все равно весь PVC.

Неудобно, но что ж поделать. Главное что ты знаешь об этом и можешь частично этим процессом руководить.

Посмотреть список PersistentVolume Claim можно такой командой:

**kubectl get pvc**



Чуть не забыл. PVC помимо объема еще характеризуются таким параметром, как "тип доступа"

Всего этих типов существует 3 штуки:

**ReadWriteOnce (RWO)** - том может принадлежать только одному Pod-у.

**ReadOnlyMany (ROM)** - том может быть подключен одновременно к нескольким Pod-ам, но только в режиме чтения.

**ReadWriteMany (RWM)** - том может быть доступен нескольким Pod-ам и на чтение и на запись.

Иииии..... есть еще одна сущность. **Storage Class.** Эта сущность идет за границами описанной выше иерархии.

Это скорее можно сравнить со справочником.

На каждый StorageClass можно задать разные политики бекапирования, приоритет обслуживания и т.д. Также можно медленные стораджи отдавать в один класс, а быстрые в другой.

Также в сущности Storage Class возможно хранить параметры подключения хранилища.

И тогда при подключении PV мы можем задать сразу его класс и тем самым применить к нему все настройки как и на других PV того же класса.

И сразу же PV добавляется в пул ресурсов определенных проектов.

Такой подход сокращает количество действий, при обслуживании кластера. А чем меньше мы делаем руками, тем лучше.

Итак. Теперь ты знаком с теорией. Давай перейдем к практике.

Давай попробуем запросить у кластера ресурсы под pvc. Т.е. создать pvc.

Для этого сделаем вот такой манифест и применим его к кластеру.

---

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: testpvc

spec:

accessModes:

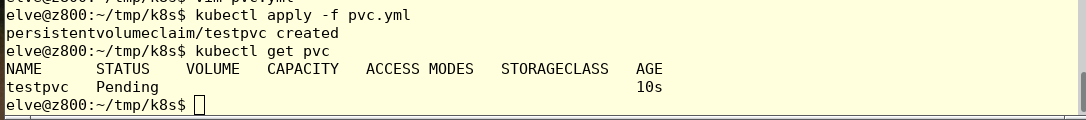
- ReadWriteMany

resources:

requests:

storage: 5Gi

И что же у нас получилось?

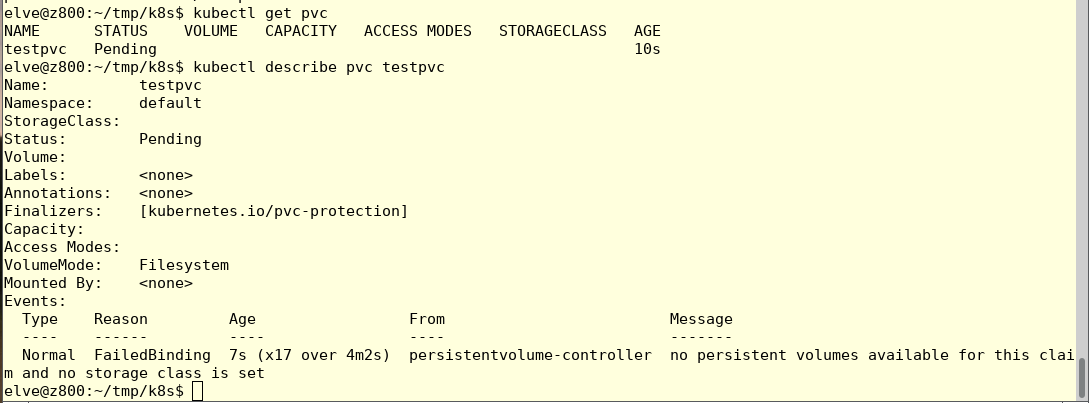


Статус pvc - "pending". Т.е. он ждет ресурсов, но кластер их не выделяет. Скорее всего их нет.

Вообще все логично. Раз не создано ни одного PV, то PVC негде брать место.

Ты можешь убедиться сам с помощью команды

**kubectl describe pvc testpvc**



Видишь в самом конце этой простыни раздел Events? Читай о чем там пишут =).

Похоже надо создать PV.

У меня нет ни дисковой полки, ни распределенной фс, вроде ceph или drbd и поэтому я тебе покажу работу с PV на базе NFS. Пусть для NFS нет провизионера, но зато и поднять такую шару проще простого в любом окружении.

Параметры моего NFS-сервера такие:

ip: 10.0.0.254

path: /vmstorage/nfs

Соответственно вот такой у меня будет конфиг для подключения:

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: my-nfs-share

spec:

capacity:

storage: 100Gi

accessModes:

- ReadWriteMany

persistentVolumeReclaimPolicy: Retain

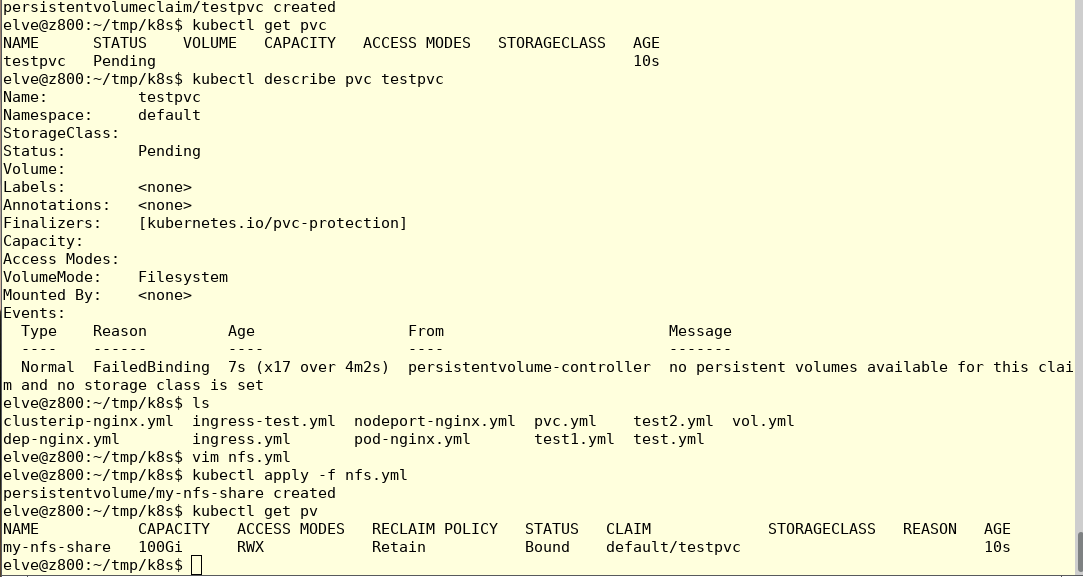
nfs:

server: 10.0.0.254

path: /mnt/vmstorage

И применяем.

И давай посмотрим что произошло с нашим pvc.



Он получается слопал весь PV на 100Гб. Как я выше и писал. Надо было 5Гб, но взял что есть и "выиграл" аж 100 =). Это к тому, что PV нужно собирать с оглядкой на такое поведение.

Но это все в условиях дефицита ресурсов. Чаще все же встречаются конфигурации с распределенными ФС, на которые есть provisioner-ы. И там каждый PVC откусывает от PV сколько ему надо. Никаких выигрышей уже нет. Одна сплошная демократия =).

Напоминаю, что на каждой ноде должна быть поддержка nfs. Если нету, то доустанови соответствующий пакет.

Для убунты и дебиана это

**apt-get install nfs-kernel-server**

Вообще ты в конфиге для PV должнен был заметить новый параметр - **persistentVolumeReclaimPolicy**.

Для чего он нужен?

Он определяет что будет с PV после удаления PVC. Есть три возможных варианта:

- **Retain** - ничего не делать.

- **Recycle** - pv будет очищен.

- **Delete** - при удалении pvc удалется еще и pv.

Итак, есть у нас pvc. Давай попробуем подключить его к Pod-у. Для этого создадим новый Pod. Пусть будет вот такая конфигурация:

---

spec:

containers:

- name: app

image: alpine

volumeMounts:

- name: data

mountPath: /mnt/nfs

command: ["/bin/sh"]

args: ["-c", "sleep 500000"]

volumes:

- name: data

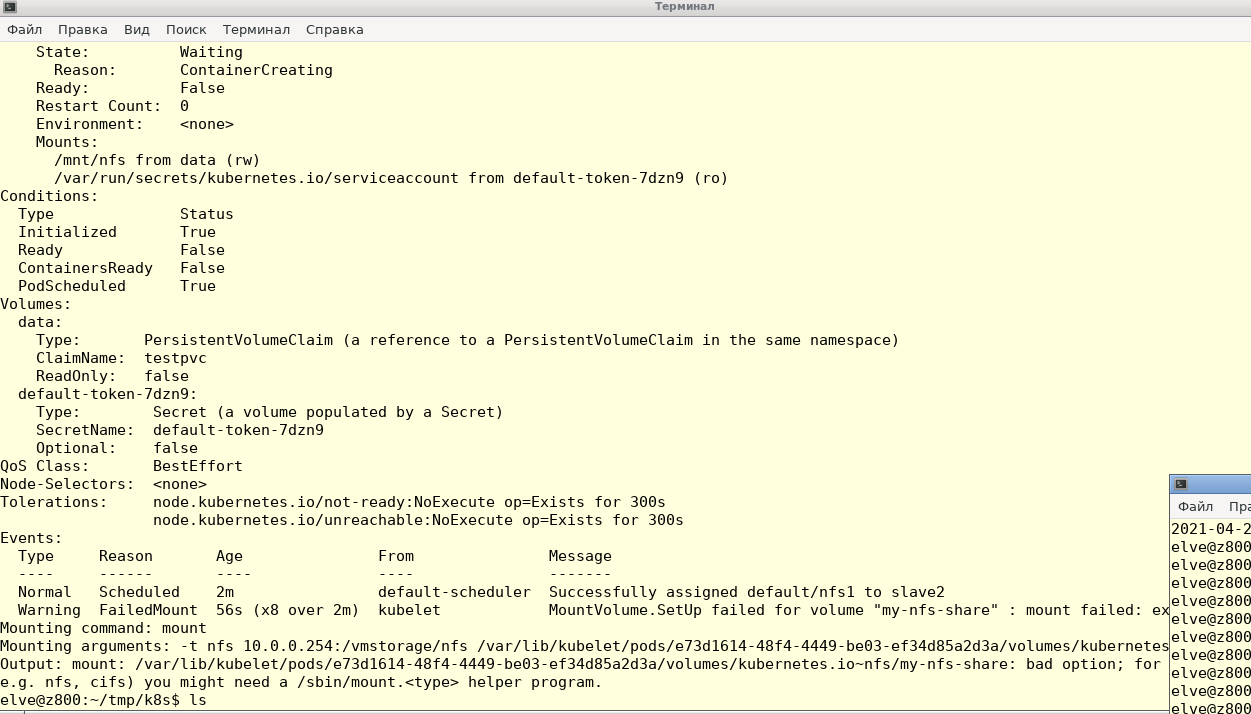
persistentVolumeClaim:

claimName: testpvc

Применяем конфиг к кластеру

**kubectl apply -f pod-nfs.yml**

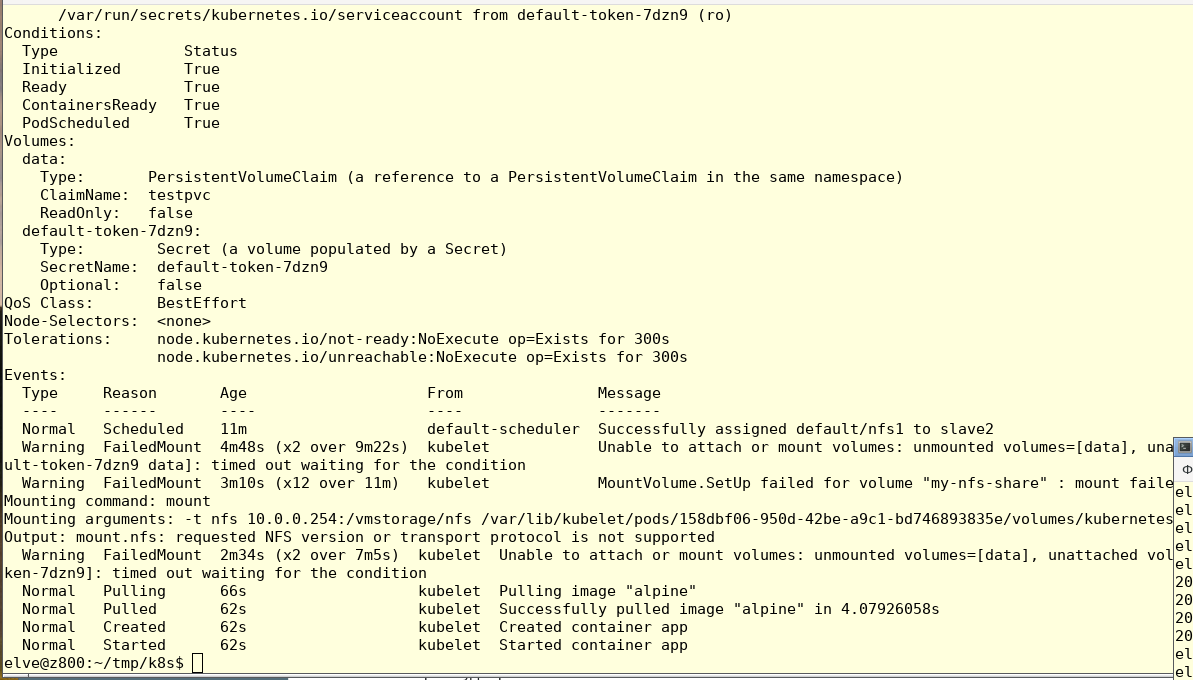
И смотрим что получилось.



А получилось то, что контейнер не создается, т.к. невозможно примонтировать nfs-шару =). После некоторых манипуляций я нашел, что у меня попросту не запущен nfs-сервер. Я его не добавил в автозагрузку и после перезагрузки сервера процесс не был поднят.

После того как я включил nfs-службу, то через пару секунд контейнер все же поднялся.

И в логах у него уже вот это:



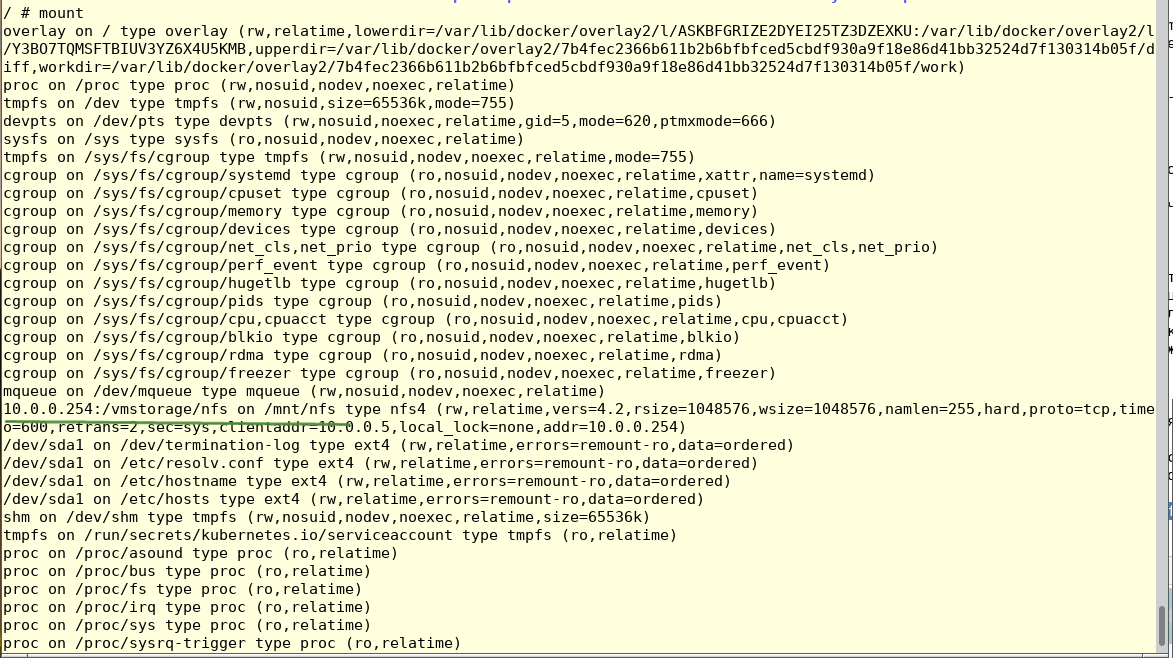
Так. Вот запустил я этот pod (и ты запустил тоже :P ) и что дальше?

Ну в целом-то у нас сейчас запущен pod с подключенным внешним хранилищем. Давай проверим, что все работает. Для этого надо зайти в консоль контейнера

**kubectl exec -it nfs1 sh**

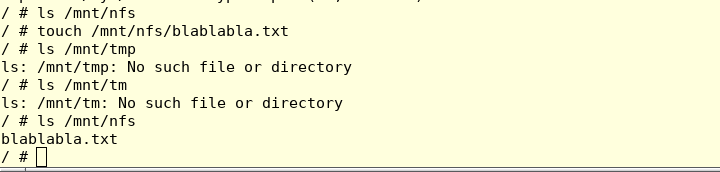
Шелл sh, т.к. bash в этом образе нет.

Итак. Мы подключились. Смотрим примонтированные устройства. nfs-шара имеется.

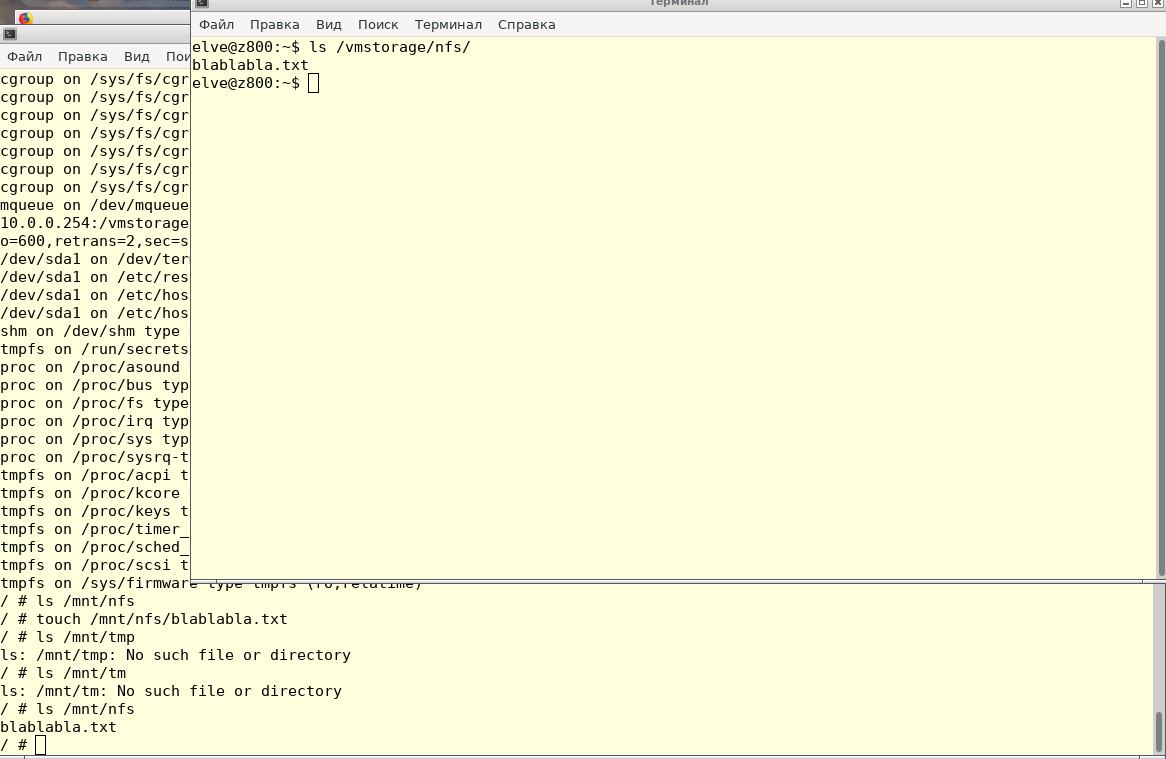


Шара пуста. Можешь в этом убедиться. Надо создать в шаре какой-то файл.

**touch /mnt/nfs/blablabla.txt**



И теперь давай проверим что файл появился и на nfs-сервере.



Круто. Т.е. общее хранилище работает.

Т.к. хранилище у нас типа ReadWriteMany, то давай для эксперимента поднимем еще один Pod. Как минимум узнаем, будут ли проблемы или все на этот раз отработает корректно =).

Вот тебе конфиг второго Pod-а.

---

apiVersion: v1

kind: Pod

metadata:

name: nfs2

spec:

containers:

- name: app2

image: alpine

volumeMounts:

- name: data

mountPath: /mnt/nfs

command: ["/bin/sh"]

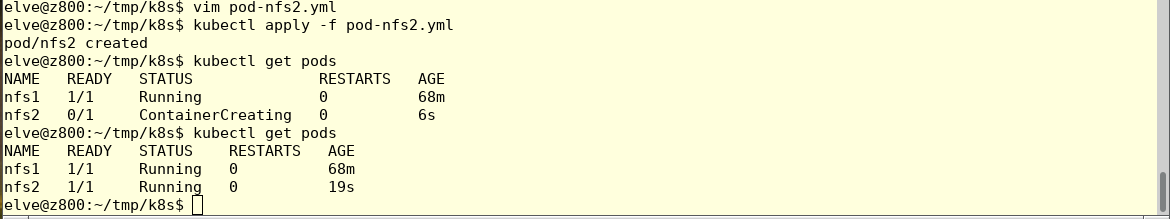
args: ["-c", "sleep 500000"]

volumes:

- name: data

persistentVolumeClaim:

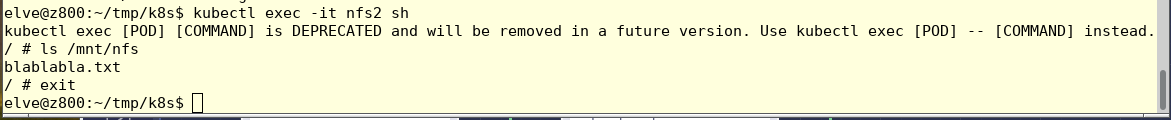
claimName: testpvc



Как видишь теперь все поднялось корректно.

Давай подключимся к второму Pod-у.

**kubectl exec -it nfs2 sh**



Как видишь, общее хранилище работает. Файл мы видим.

В общем-то все. Так примерно и работают с хранилищами в кластере.

Если все что мы тут насоздавали больше не нужно, то можно все поудалять.

**kubectl delete pod nfs1**

**kubectl delete pod nfs2**

**kubectl delete pvc testpvc**

**kubectl delete pv my-nfs-share**

В принципе все остальное ты можешь и сам узнать, основываясь на абстракциях, которые узнал в этом уроке. Но я еще покажу тебе пример создания локальных PV. Локальные - значит создаются прямо на нодах. Фактически, ты просто шаришь папку на ноде для контейнеров =).

Т.к. это уже вторая задача по хранилищам, то давай сделаем все как взрослые специалисты со всеми абстракциями.

Сначала создадим StorageClass, потом создадим PV и укажем наш SC. Ну а дальше уже PVC и тестовый Pod (или несколько, если захочешь). Порядок создания именно такой, т.к. одно зависит от другого.

Итак. Storage Class

---

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: local-storage

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer

Применяем.

**kubectl apply -f sc.yml**

Теперь PV. Я сделал для обеих нод. Мне так интереснее. Ты лучше тоже так сделай ).

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-local-slave-1

spec:

capacity:

storage: 10Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain

storageClassName: local-storage

local:

path: /mnt/local-storage

nodeAffinity:

required:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

- slave1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-local-slave-2

spec:

capacity:

storage: 10Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain

storageClassName: local-storage

local:

path: /mnt/local-storage

nodeAffinity:

required:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

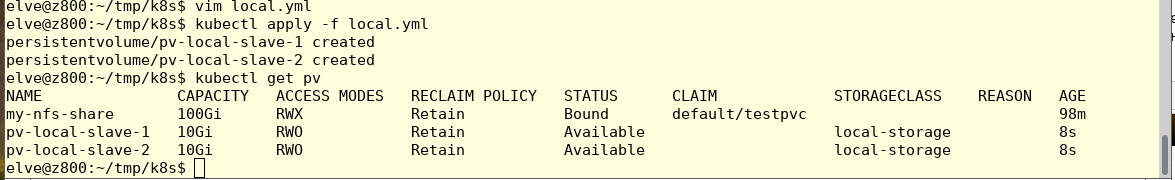
- slave2

И применяем конфиг PV

**kubectl apply -f local.yml**

Ты обратил внимание на значение **path**? Если нет, то я уже тебе об этом пишу. Путь, который там указан, должен на нодах существовать. Поэтому заходи и создавай папки перед тем как создавать PV-шки.

Вот так вот у меня выглядит список PersistentVolume. Я не удалял PV с nfs =).



Теперь возьмем класс local-storage и навесим на него PVC.

Ты мог бы сделать по одному PVC на каждый PV . Это ты уже умеешь. Но лучше делай как я говорю - это будет интересно =)

---

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: local-volume

spec:

storageClassName: "local-storage"

accessModes:

- ReadWriteOnce

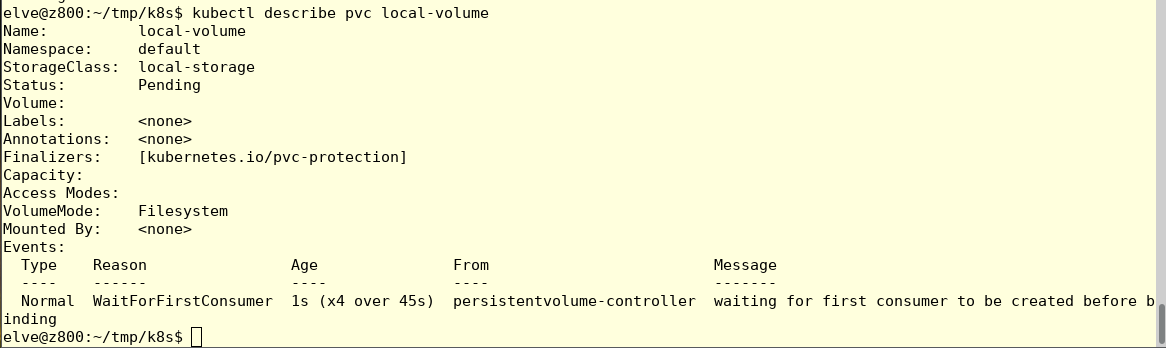
resources:

requests:

storage: 10Gi

Применяем

**kubectl apply -f local-pvc.yml**



Это не ошибка. Ты обратил внимание, что pvc создан с указанием не на конкретный pv, а на StorageClass. У нас два PV с этим классом. Как только на одной из нод будет запущен Pod - ее хранилище и уйдет к PVC. С одной стороны удобство - с другой стороны... хранилища этого класса на других нодах будут простаивать.

Создадим Pod вот такой конфигурации:

---

apiVersion: v1

kind: Pod

metadata:

name: localpod1

spec:

containers:

- name: app

image: alpine

volumeMounts:

- name: data

mountPath: /mnt/nfs

command: ["/bin/sh"]

args: ["-c", "sleep 500000"]

volumes:

- name: data

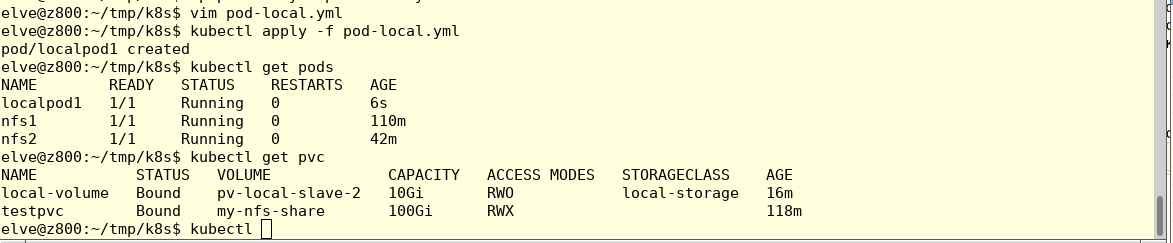
persistentVolumeClaim:

claimName: local-volume

Применняем...

**kubectl apply -f pod-local.yml**

И не будем смотреть на состояние Pod-а. А сразу посмотрим что там с нашим PVC



Как видишь, теперь он определился откуда брать PV.

Вот теперь точно все по подключаемым хранилищам. Этого тебе на первое время должно хватить.

Впереди еще много интересного. А эта глава закончена