Обзор подсистем хранения данных в Kubernetes. CSI

Не забудь включить запись!

План

- Немного концептуальных вещей
- Volumes
- Persistent Volumes, Storage Classes, Dynamic Provisioning
- CSI
- Bonus! Разбор хранения - блочное устройство, файловая система

Концепция

Что мы хотим от подсистемы хранения?

Условно, это можно разделить на:

- хранение самой операционной системы
- хранение данных приложений

> Эти вещи различаются, потому что к ним различны подходы

Хранение в кластере

- Хранение самое операционной системы неважно
- Настройки операционной системы либо неважны
  - Сервисы позволяют нам не париться про конкретный IP-адрес какого-то контейнера
- Либо описаны декларативно нами в наших yaml-файлах
- Настройки операционной системы не меняются более вручную
  - Хочется в это верить 🥰
- `etckeeper` больше не нужен, у нас есть `etcd`

Хранение в кластере

- Важная мысль: контейнеры по сути своей сами не хранят ничего
  - Но мы можем прикреплять к ним разные виды хранилищ

  А как про приложения?

- Как только мы ушли в микросервисы, мы получили определенную изоляцию контейнеров друг от друга
- Однако порой им тоже нужно обмениваться файлами порой

Облако или bare-metal

- Облачные решения устроены так, что Вам не нужно более думать о дисках или отказоустойчивости
- Вы даже сможете выбирать, насколько производительные диски взять
- Вопрос горизонтального масштабирования решается простым изменением размера или добавлением еще диска

Но будут и минусы

- Будет ряд случаев, когда latency окажется неприемлемой
- Либо в результате роста объема своя система хранения данных окажется выгоднее

А что такого особенного в подсистеме хранения?

- С одной стороны, нет большой разницы, прочитать ли файл за 1 мс или за 10 мс
  - Основная latency будет лежать не у Вас в приложении, а в том как к Вам идет пользователь
  - Но есть разница в 10 мс или 100 мс
- С другой стороны, с ростом объемов невысокая производительность облачных решений может стать проблемой

Volumes

Вместо тысячи слов для начала

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: test
spec:
  containers:
    - name: test
      image: ubuntu
      command: ["/bin/bash", "-ec", "while :; do sleep 2; done"]
      volumeMounts:          # а что вообще надо монтировать?
      - mountPath: /test     # куда монтирую?
        name: test-volume    # что монтирую?
  volumes:
  - name: test-volume
    emptyDir: {}             # вид хранилища с параметрами
  restartPolicy: Never
```


VOLUMES
imgflip.com

Что мы поняли из этого примера?

- Volume описывается на уровне Pods, ссылка идет на уровне контейнера
- Хранилище монтируется, то есть чаще всего речь будет идти о доступе к директории, не к блочному устройству
    - Отдельный интересный вопрос - опции монтирования

Огласите весь список, пожалуйста

- Нативное
- Облачное
- Семья ceph
- [Кровавый] Enterprise 
- Сетевые ФС
- Новинки
- Расширения

Нативное

- emptyDir
- hostPath
- local
- configMap
- secret
- downwardAPI
- projected
- gitRepo (deprecated)

Облачное

- `awsElasticBlockStore`
- `azureDisk` - аттачится к одной машине, другим нельзя
- `azureFile` - другой интерфейс для общения, позволяет обращаться с других машин

> Подробнее про разницу в Azure

- `gcePersistentDisk` - изначально работает в логике "множественное чтение"

Семья ceph

- `rbd` - Rados Block Device
- `cephfs` - CephFS

[Кровавый] Enterprise 

- `iscsi`
- `fc`
- `cinder` - OpenStack Cinder Volumes
- `vsphereVolume` - VMware
- `scaleIO` - Dell EMC → VxFlex OS

Сетевые ФС

- `glusterfs`
- `nfs`

> Эти сетевые файловые системы здесь уже достаточно давно, хорошо изучены. Считаются более зрелыми, нежели Ceph FS

Новинки

- `flocker`
- `portworxVolume`
- `quobyte`
- `storageos`

> Это разработки, которые появились вместе с k8s, оптимизированы под него, обладают рядом специфичных фич

Расширения

- `flexVolume`
  - Позволяет подключать любое хранилище при помощи исполняемого "коннектора"
- `csi`
  - Стандартизованный интерфейс для создания драйверов хранилища

subPath

- Мы можем монтировать не корневую директорию Volume, а некую произвольную
- Тогда при помощи mountPath/subPath можем пояснить, что конкретно

```
volumeMounts:
- mountPath: /var/lib/mysql
  name: site-data
  subPath: mysql
```

[Demo 01]

Volumes

- Просто, но управлять ими надо вручную, в spec для Pod
- Требуют познаний о том, как устроена инфраструктура
    - Это страшное знание каких-то ID и вообще
- Не позволяют в полной мере абстрагироваться (привязка к виду хранилища)
- Имеют разнообразный жизненный цикл (что-то размонтируется, что-то чистится)

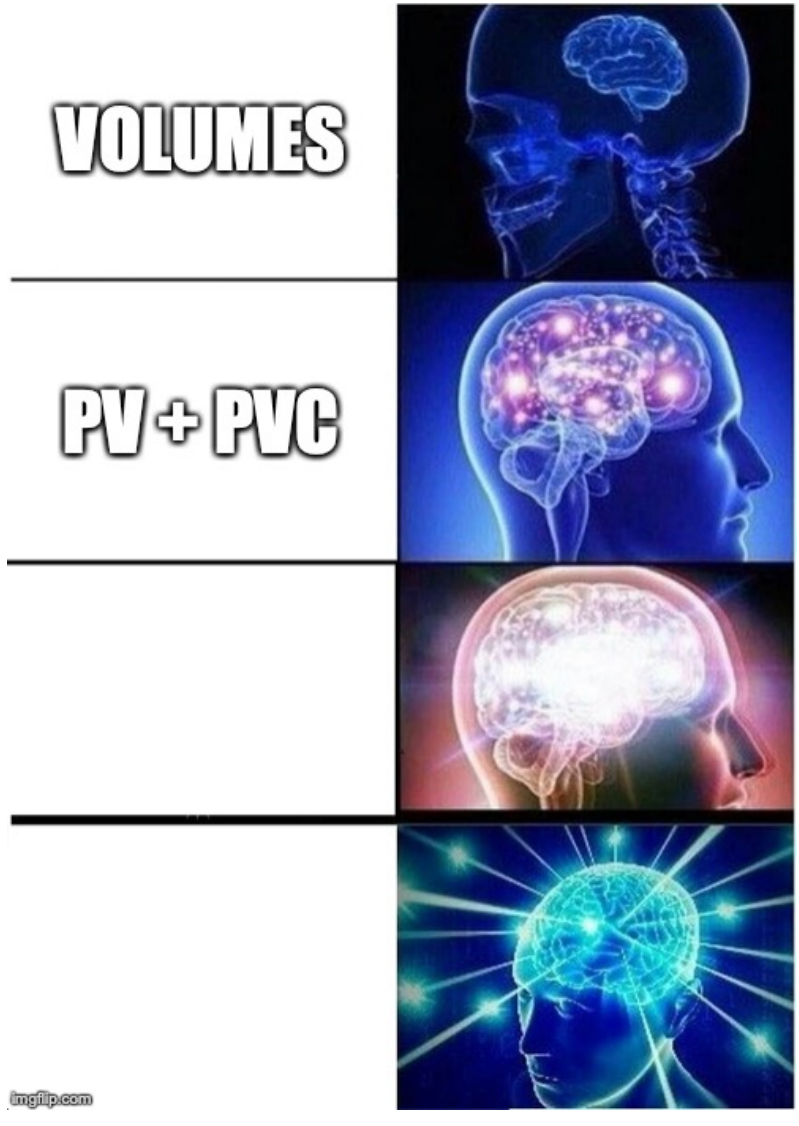
Когда хорошо использовать Volumes?

- Очень простые пожелания от хранилища
- Всё хранилище одинаковое по требованиям скорости
- Нет динамического создания Pod с разными данными
    - Все Pod используют только для чтения один Volume, например

Persistent Volumes, Storage Classes, Dynamic Provisioning

Встречаем PersistentVolume

- Без них
  - Pod ➜ Volume (DevOps) ➜ [Disk]
- С ним
  - Pod ➜ PersistentVolumeClaim (Dev) ➜ PersistentVolume (Ops) ➜ [Disk]
- PersistentVolume = описание единицы хранилища (с подробностями)
  - PersistentVolumeClaim = описание хотелок от него (но без подробностей)


VOLUMES
PV + PVC
imgflip.com

PersistentVolume

- Связка между PV и PVC - один к одному
  - Но вот один PVC можно сунуть в несколько Pods
- Частая ошибка - считать, что к одному PersistentVolume можно подсунуть много PersistentVolumeClaim
- Разработчик меньше знает о том, как устроено хранилище
  - Он описывает только PVC, а вот создавать PV - удел администратора

> Не все плагины хранилища могут PV! Полезная табличка про Storage

Как k8s связывает PVC и PV?

- Параметр storageClassName (и у PV, и у PVC)
  - `storageClassName: local-storage`
- Можно использовать selector

```
selector:
matchLabels:
  release: "stable"
matchExpressions:
  - {key: environment, operator: In, values: [dev]}
```

Как k8s связывает PVC и PV?

- Важно! storageClassName можно не указать, либо указать пустым ""
- Это приведет к двум **разным** последствиям
- Кстати, применение selector тоже, но об этом чуть позже

[Demo 02]

Связал? Развяжи!

- Как насчет высвобождения PV?
  - Падение контейнера, кстати, не повод рушить связку PVC ➜ PV
  - Потому что PVC по сути самостоятельный объект
- Как только связка PVC ➜ PV установлена, она типа "навечно"
- Но если мы вдруг удалим PVC, то дальше в ход пойдет параметр `persistentVolumeReclaimPolicy`

Развяжи, а не стреляй в ногу

- Нельзя удалить PVC, на которую ссылается Pod
- Нельзя удалить PV, на которую ссылается PVC
- Это сделано при помощи механизма Finalizers
  - Набор хуков, коотрые выполняются до удаления

[Demo 03 - Bonus Track]

Удаляй меня скорей

- Когда удаляется PVC, нужно решить, что делать с PV
  - Напомню, что связка у нас один-к-одному
- `persistentVolumeReclaimPolicy`
  - Retain - не трогать
  - Delete - удалить PV + объект в облаке
  - Recycle - просто почистить содержимое (deprecated)

mountOptions

- Можно указать для PV особые опции монтирования
- Их поддерживают не все плагины

Полезная табличка про Storage

Что мы поняли?

- Пока мы говорили о строго статическом управлении хранилищем
- Это означает, что все используемые нами PV до этого надо с любовью создать
- В несложной среде можно этим позаниматься, да

hostPath или local?

Лирическое отступление

hostPath

- "Наивный" способ прокинуть что-либо внутрь Pod - устройство, сокет, файл, директорию
- Никакой поддержки PersistentVolume в нормальном кластере (только с одной нодой)
- А поскольку нет PV, то нет и nodeAffinity
- Поэтому назначать будем ручками

local

- Поддерживает PersistentVolume, а значит, возможность делать nodeAffinity
- Предназначен для Storage - "прокидывается" блочка или файловая система

Давайте уволим администратора?

- Мы до этого называли вещи немного не своими именами
- Администратор делал для нас то, что называется "provisioning"
- И удивительно, но в облаке это можно делать автоматически (Dynamic Provisioning)

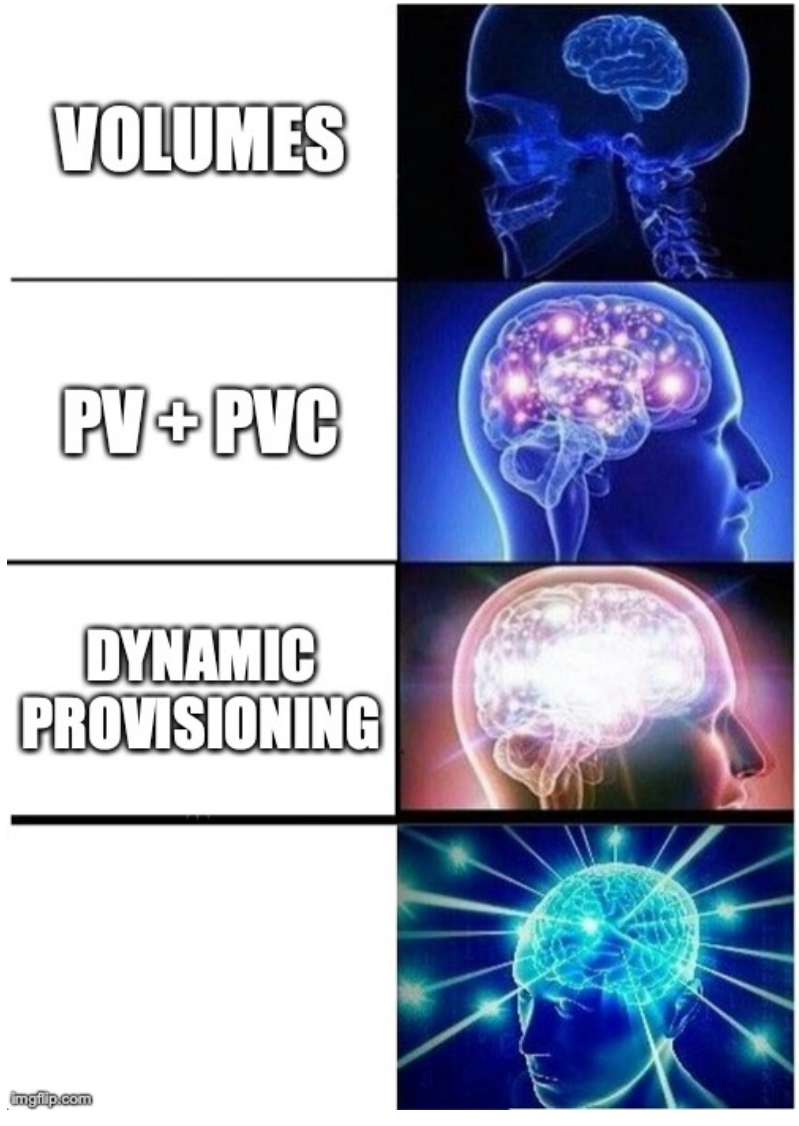

ВОТ ЭТО
ПОВОРОТ!!

Встречаем объект StorageClass

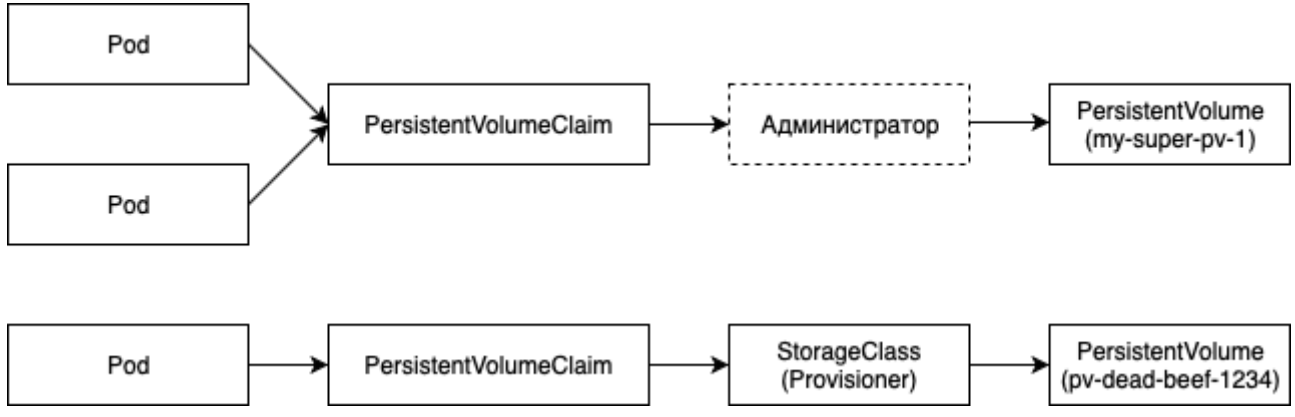
- Выполняет связку между некоторым именем (fast, slow)
- И способом его создания (GCP, etc)
- С параметрами (pd-standard, pd-ssd)

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
```

[Demo 04]


VOLUMES
PV + PVC
DYNAMIC PROVISIONING
imgflip.com

Немножко схематики



Когда Dynamic Provisioning (DP) не сработает?

- Если PVC содержит selector, то никакого DP
- Если PVC запрашивает storageClassName: "" (именно **пустой**), то никакого DP
- Вообще за DP отвечает Admission Controller по имени DefaultStorageClass
  - При **неуказанном** storageClassName он подставит StorageClass с пометкой default
  - Стоит ли говорить, что этот Admission Controller должен быть включен? 🙃

Воу-воу, минуточку

- По сути своей StorageClass должен сатисфачить запросы на Storage сразу же
  - Создал PVC? Получи же скорее Storage! И неважно, что Pod еще пока не существует
  - Это может создавать конфликт, когда Storage и Pod зашедулились в разных зонах, а драйвер Storage не умеет так жить
  - Или scheduler сделал PV → PVC на какой-то ноде, а ресурсов не хватило стартовать там Pod

Воу-воу, минуточку

- Порой это не очень удобно, поэтому у StorageClass есть параметр volumeBindingMode
  - Можно установить его в WaitForFirstConsumer, чтобы делать provisioning только когда появилось, кому

> WaitForFirstConsumer не всем доступен! Полезная табличка про Storage

StorageClass для Local?

- Local вообще не поддерживает DP, а PV поддерживает только в Single node
- Зачем тогда создавать для него StorageClass?
  - Это можно найти во множестве статей и примеров
- Ответ: чтобы использовать volumeBindingMode: WaitForFirstConsumer
- Тогда scheduler сможет сначала решить, можно ли там стартовать, а дальше и PVC к PV правильно приклеит

Изменение размера PVC

- Только в сторону увеличения
- Только для DP (свойство allowVolumeExpansion: true в StorageClass)
- Есть еще PersistentVolumeClaimResize Admission Controller
  - Запрещает изменение размера для тех, кто это не поддерживает
  - StorageClass должен явно задавать allowVolumeExpansion: true
- Flex и CSI можно изменить размер только если это поддерживается

Как изменить размер PVC

- Да просто отредактировать PVC, указав новый размер
- Не будет создаваться новый PV, будет изменен размер существующего
- Произойдет это в момент рестарта Pod (но нужен доступ ReadWrite, разумеется)
    - Есть возможность ресайза без рестарта (beta в 1.15)
    - Но нужно чтобы этот PVC был связан с Pod

CSI

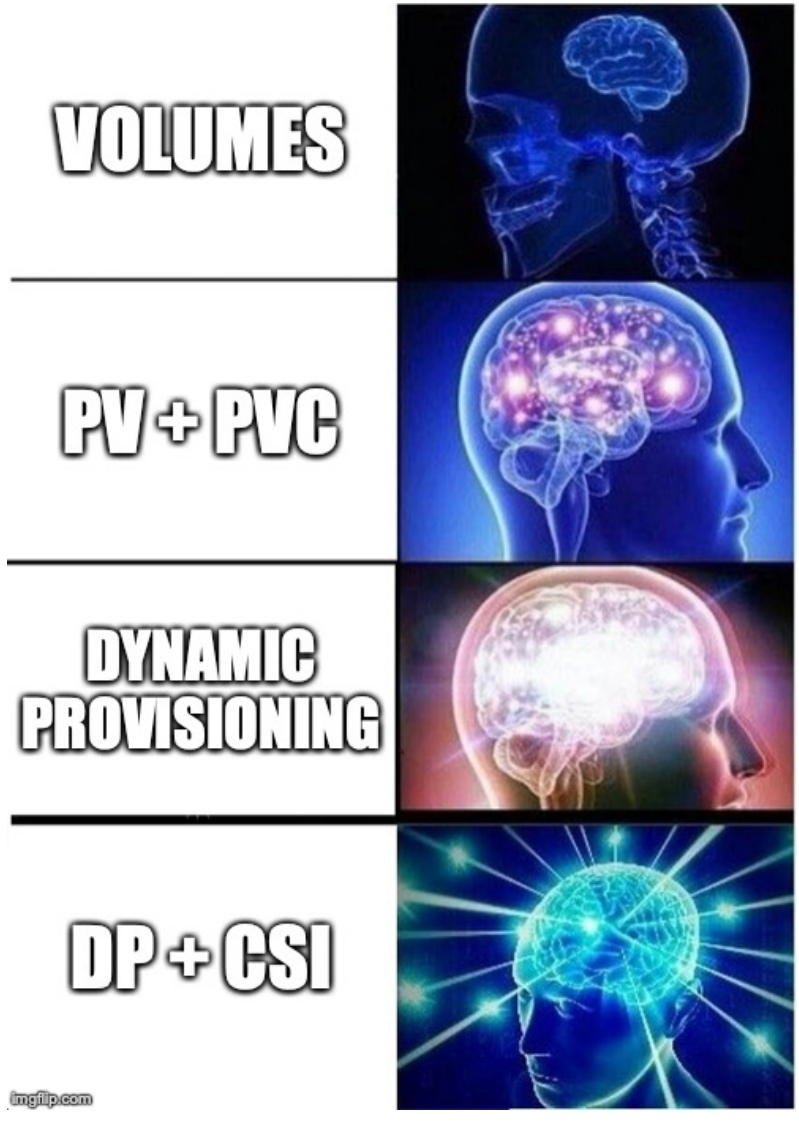
Поговорим про CSI

- Проблема плагинов в k8s для Storage в том, что циклы релизов разные
- Условно, производитель технологии может обновляться реже/чаще, нежели k8s
- Поэтому понадобилась архитектура, позволяющая добавлять в k8s свой драйвер Storage
- Она-то и зовется Container Storage Interface (CSI)

Хорошо, зачем мне это?

- CSI по функционалу будет развиваться быстрее
- Там уже 47 драйверов, будет, разумеется, больше
- Две нужные фичи:
  - Volume Snapshots (10 раз Yes, остальные либо No, либо 🤷)
  - Volume Cloning (5 раз No и 42 раза 🤷)

> Посмотеть на список драйверов можно вот здесь


VOLUMES
PV + PVC
DYNAMIC PROVISIONING
DP + CSI
imgflip.com

Что мне нужно включить для CSI?

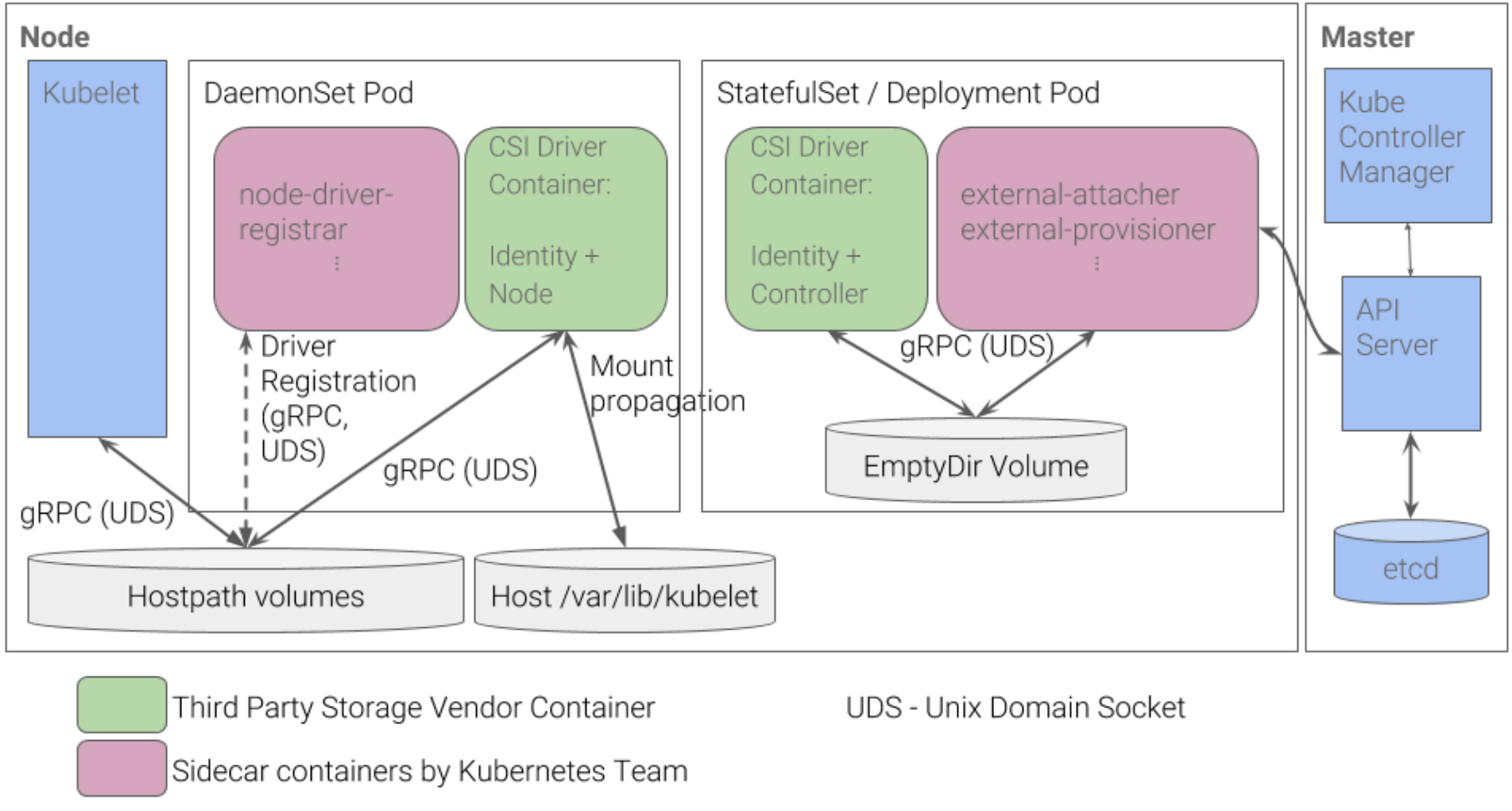
- Ну, во-первых, нужно разрешить privileged Pods (сейчас это и так есть)
- Mount Propagation
  - `MountFlags=shared` в .service, если запуск при помощи systemd

Как мне это установить?

- Обычно производитель CSI драйвера дает репозиторий с кодом деплоя
- Результатом установки будет создание набора Pods, которые реализуют нужные фичи
- PROFIT!

Рекомендации по деплою CSI драйвера

Простенькая архитектура



Чуть про драйвер

- Понятие "драйвер" для CSI по сути не то, что драйвер в обычном смысле (для ОС)
- Он выполняет некую "стыковку" пожеланий хранилища и самого хранилища
- Не нужно думать, что через него идут все операции ввода-вывода, это не так

[Demo 05]

И что там внутри?

- `cluster-driver-registrar` - создает объект CSIDriver в кластере
- `node-driver-registrar` - объясняет kubelet, где находится socket, через который общаться с драйвером
- `external-provisioner` - создание/удаление

И что там внутри?

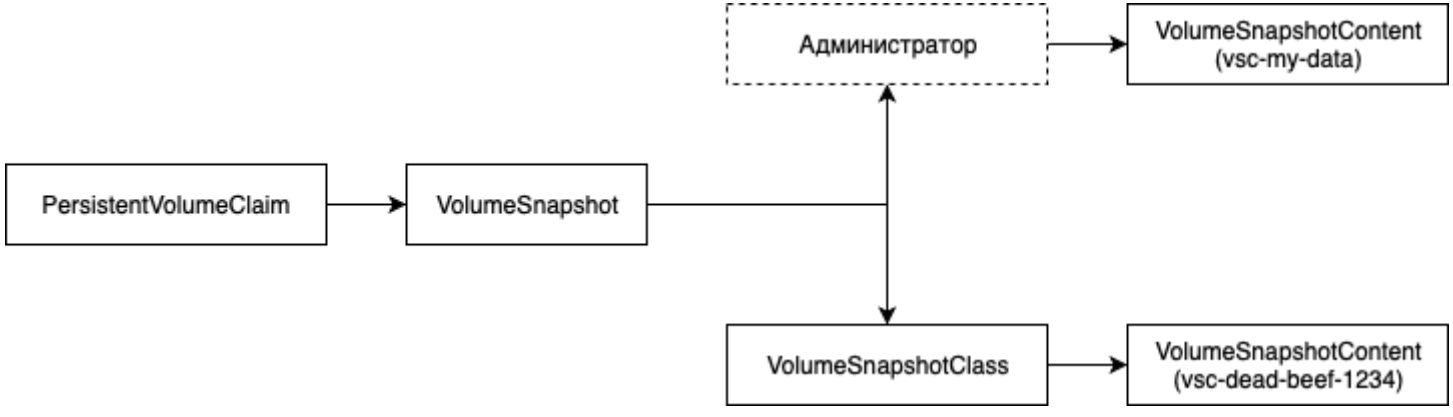
- `external-attacher` - выполняет attach/detach
- `external-resizer` - изменение размеров
- `external-snapshotter` - снапшоты
- `livenessprobe` - тыкает палочкой на предмет живости

Volume Snapshot

- Это все еще alpha-фича для 1.15, поэтому `--feature-gates="VolumeSnapshotDataSource=true"`
- Только с CSI, ни с чем другим не работает
- CSI драйвер должен это поддерживать

[Demo 06]

Немножко схематики



Volume Cloning

- Это alpha-фича для 1.15, поэтому `--feature-gates="VolumePVCDataSource=true"`
- Только с CSI, ни с чем другим не работает
- CSI драйвер должен это поддерживать
- Только в рамках одного Namespace (источник и назначение)
- Клонирование возможно только для Dynamic Provisioning

Volume Cloning

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: clone-of-pvc-1
  namespace: myns
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: pvc-1
```

Рекомендации

Облака

- Если пошли туда, то до конца
- Попытка сочетать облако и локальное хранение обычно такая себе идея
- Большая часть дисков уже имеет встроенную отказоустойчивость
- Снапшоты позволяют достаточно удобно делать резервные копии

Bare metal - уровень Normal

- Мы держим на нодах диски, на которых можно хранить данные
  - Используем local + nodeAffinity, чтобы знать, куда что приземляется
- Используем LVM, чтобы делать снапшоты
  - Бекап делается со снапшота
  - Снапшот рушит производительность, поэтому иногда сложнее
  - Раздел → Снапшот → Копия раздела → Резервная копия

Bare metal - уровень Normal

- Резервные копии храним на других нодах
- Этим достаточно муторно управлять, но если нод 3-5 штук, то вполне можно
- Снапшоты не помогут с БД, поскольку там не всегда понятно, записались ли данные
    - Либо лочим базу, либо снимаем с нее дамп

Bare metal - уровень Nightmare

- Приходим к мысли, что единая полка или парочка - хорошая мысль
- Делаем отдельную сеть для стораджа и раздаем с полки LVM по iSCSI
    - Делаем локальные копии с помощью снапшотов, но тут уже возможны нюансы
- Дешево, сердито, быстро
- Единая точка отказа

Bare metal - уровень Hell

- Раз у нас кластер для Compute, почему бы и не кластер для Storage?
- ceph + RBD замечательно работает (где-то можно даже чуть CephFS, если скорость позволяет)
- Вдумчивое управление кластером ceph достаточно хитрое колдунство
- Но широкие возможности руления redundancy и вот этим всем

Bare metal - Secret Cow Level

- И я не говорю, что нельзя, допустим, внутри k8s катать ceph или glusterfs
- Я просто пока считаю, что с этим будет много нюансов
- Но с другой стороны, это вполне имеет право на жизнь

Разбор хранения

Кладовщик Михалыч

- Он работает на складе, где все полки пронумерованы
- Умеет делать простые две вещи
  - Привезти ящик с полки
  - Положить ящик на полку обратно
- Вся его работа заключается в этом
- Он не знает, что находится в ящике. И не хочет.

Завскладом Зинаида

- Знает, что лежит в ящиках
- Умеет сопоставлять содержимое ящиков между собой
  - Чтобы замешать тесто, нужно содержимое нескольких ящиков
  - Количество теста в итоге есть некая функция от разных ящиков

Михалыч - блочное устройство

- Может притащить ящик целиком
- Не может "докинуть" в ящик мешок муки
  - Только притащить ящик с полки, положить мешок, обратно ящик на полку

Зинаида - файловая система

- В разные ящики она может раскладывать разные компоненты
  - Ее мнение о размере ящика может не совпадать с мнением Михалыча
- В одном ящике может лежать и мука, и дрожжи - пока Зинаида об этом знает
- Обычно одно движение для Зинаиды означает кучу работы для Михалыча
  - Зинаида добавляет на склад сырья на 50 булочек = 5 ящиков муки, 1 ящик дрожжей, 1 ящик сахара

Атомарность

- Михалыч атомарен по природе своей - один ящик, принес, унес
  - Недостача ящиков для него не особо проблема
- Зинаида по природе своей хотела бы тоже такой быть, но нет
  - Большинство ее операций касается нескольких ящиков
  - Недостача одного важного ящика может остановить всё производство наглухо

А зачем мне это?

- k8s пока больше ориентирован на использование файловых систем
- Но есть приложения, которым нужно давать в подчинение блочные устройства
  - ceph, например
- Хранилище в k8s пока всё больше про файловые системы
  - Которые по сути своей могут сильно по-разному себя вести

Что значит "по-разному"?

- CAP-теорема в чистом виде
  - "Быстро, удобно, дешево" - выбери любые два качества
- Не бывает чудес - любая распределенная файловая система (GlusterFS) или работающая с множеством клиентов (NFS) будет испытывать сложности с записью
  - Потому что запись по своей сути не одна операция, а три, идущих вместе