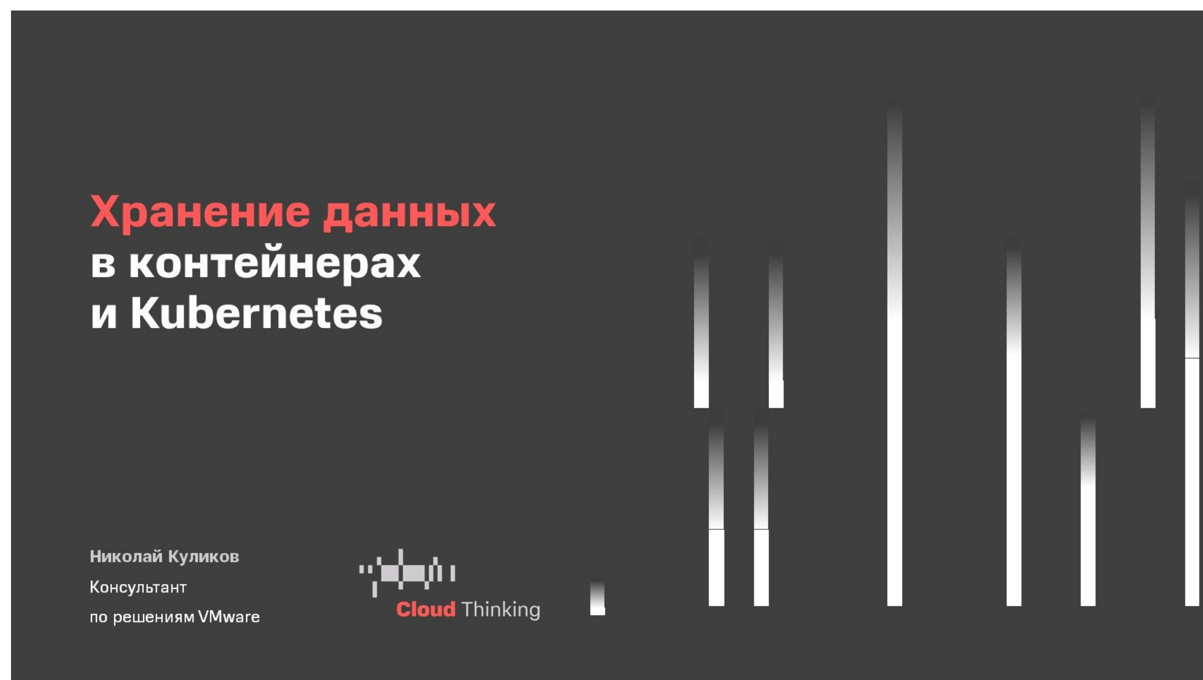
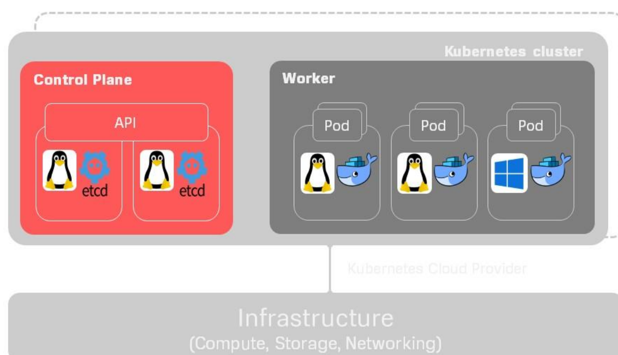
Модуль 3, урок 2



Продолжаем говорить о хранении данных в контейнерах и Kubernetes. Начнем с базовых вещей.



Дополнительный слой сверху, который занимается оркестрацией (управлением и автоматизацией) запуска и работы контейнеров

Некоторые особенности K8s:
Управление путем указания целевого состояния
Целевое состояние хранится в etcd
Всё взаимодействие через API-сервер
Расширяемость и интеграция через API

Kubernetes — это оркестратор контейнеров.
Зачем он нужен? Управлять большим количеством контейнеров вручную неудобно. Чтобы упростить процесс, требуется некая система, которая возьмет на себя автоматизацию и оркестрацию контейнеров.

Сегодня Kubernetes считается отраслевым стандартом оркестрации. Конечно, существуют и другие системы (например, Docker Swarm), но на данный момент они заняли место нишевых инструментов.

Ключевые особенности Kubernetes

У Kubernetes есть control plane — база на трех инстансах в продуктиве для отказоустойчивости. Некий API Server, через который происходит взаимодействие. Одна из ключевых концепций Kubernetes — desired state управления, когда мы задаем некое целевое состояние, то есть говорим, какими мы хотим видеть запущенные контейнеры в инфраструктуре. Целевое состояние хранится в etcd. Kubernetes сверяет текущее и целевое состояния. Когда что-то идет не так (например, у вас должно быть два контейнера, но один упал), он приводит текущее состояние к целевому — в нашем примере запускает недостающий контейнер.

В Kubernetes изначально заложена большая расширяемость и интеграция через набор стандартных API. Мы будем говорить об API, которые занимаются взаимодействием с инфраструктурой, а именно:

- запуск и получение вычислительных ресурсов;
- взаимодействие с сетями (CNI);
- взаимодействие с системой хранения (CSI).

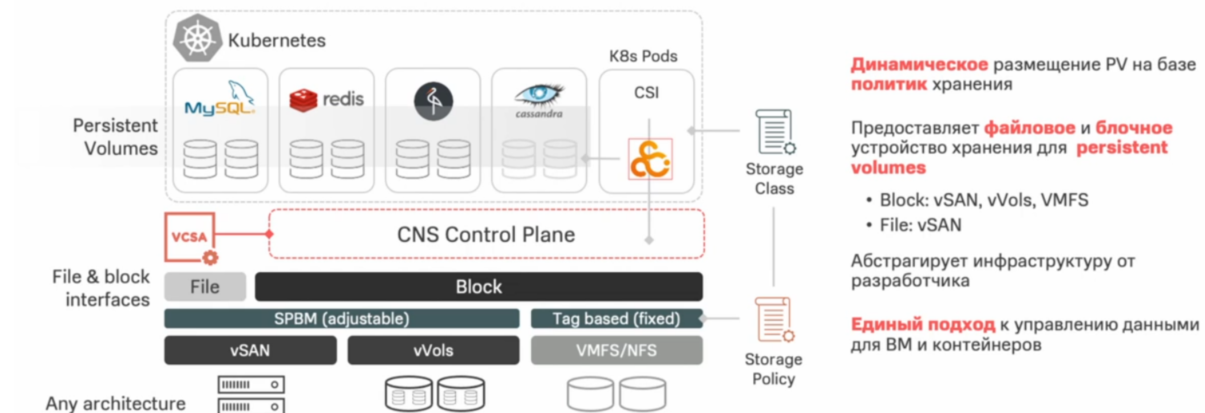
CSI — основной способ взаимодействия Kubernetes с инфраструктурой для обеспечения базового функционала работы с хранилищами: запрос ресурсов и дополнительные операции с ними (расширение, клоны, снэпы и пр.).

Проект CSI появился не так давно. Изначально он был интегрирован внутрь Kubernetes. Однако это было неудобно, поскольку вносило инфраструктурно зависимый код в кодовую базу самого Kubernetes.

Решает эту проблему стандартное взаимодействие через API, по которому Kubernetes может общаться с любыми внешними хранилищами и делать туда вызовы.

Далее мы будем разбирать особенности хранения данных на примере работы с VMware и vSphere CSI — реализацией драйвера от VMware.

Cloud Native Storage в VMware



Как эта интеграция выглядит с точки зрения VMware.
В Kubernetes запущены некие контейнеры и приложения. CSI взаимодействует с инфраструктурой. Инфраструктура представляет собой кусочек платформы управления гипервизором VMware vCenter — Cloud Native Storage. Это control plane: нет данных, только управление. CSI-драйвер общается с компонентом CNS внутри vSphere.

Все, что происходит ниже, — стандартная история с точки зрения VMware. Предоставляется блочное устройство хранения, которое может быть на базе любой системы хранения, подключенной к VMware (VMFS, NFS, vVols, vSAN). Иными словами, слой унификации абстрагирует инфраструктуру от разработчиков и позволяет им делать стандартные вызовы для получения блочных ресурсов хранения нужного размера.

Общая инфраструктура: на платформе одинаковым образом работают и контейнеры, и виртуальные машины.

CSI-драйвер имеет набор поддерживаемого функционала, то есть API-вызовов, которые он может делать в сторону инфраструктуры. Их набор меняется в зависимости от:

- версии Kubernetes (появляется новый функционал);
- уровня платформы и драйвера.

Пример. *Таблица ниже демонстрирует, что какой-то базовый функционал присутствует во всех версиях с момента его появления в 6.7U3, а что-то появляется только в последних релизах.*

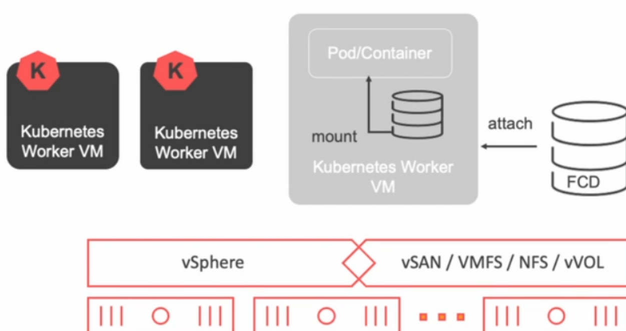
Feature	Category	Supported vSphere CSI Driver Releases	vSphere 7.0u2	vSphere 7.0u1	vSphere 7.0	vSphere 6.7U3
Enhanced Object Health in UI for vSAN Datastores	GA	v2.0.0 to v2.3.0	✅	✅	✅	✅
Dynamic Block PV support (`Read-Write-Once` Access Mode)	GA	v2.0.0 to v2.3.0	✅	✅	✅	✅
Dynamic Virtual Volume (vVOL) PV support	GA	v2.0.0 to v2.3.0	✅	✅	✅	✅
Topology/Availability Zone support (Block Volume only)	Beta	v2.0.0 to v2.3.0	✅	✅	✅	✅
Static PV Provisioning	GA	v2.0.0 to v2.3.0	✅	✅	✅	✅
K8s Multi-node Control Plane support	GA	v2.0.0 to v2.3.0	✅	✅	✅	✅
WaitForFirstConsumer	Beta	v2.0.0 to v2.3.0	✅	✅	✅	✅

Feature		Supported vSphere CSI Driver Releases	vSphere 7.0u2	vSphere 7.0u1	vSphere 7.0	vSphere 6.7U3
Offline Volume Expansion support (Block Volume only)	Beta	v2.0.0 to v2.3.0	✅	✅	✅	❌
Encryption support via VMcrypt (Block Volume only)	GA	v2.0.0 to v2.3.0	✅	✅	✅	❌
Dynamic File PV support through vSAN 7.0 File Services on vSAN Datastores	GA	v2.0.0 to v2.3.0	✅	✅	✅	❌
In-tree vSphere volume migration to CSI	Beta	v2.1.0 to v2.3.0	✅	✅	❌	❌
Online Volume Expansion support (Block Volume only)	Beta	v2.2.0 to v2.3.0	✅	❌	❌	❌
XFS Filesystem support	Alpha	v2.3.0	✅	✅	✅	✅
Raw Block Volume support	Alpha	v2.3.0	✅	✅	✅	✅

Стандартизация API позволяет работать с любыми дистрибутивами. Вы можете запустить любой дистрибутив Kubernetes поверх платформы и интегрировать CSI-драйвер.

Хранение данных в контейнерах

Когда Kubernetes-кластер развернут в виде ВМ на vSphere



K8s в формате ВМ поверх vSphere:

Создается FCD/IVD диск в vSphere

Нужно подключить **диск, который будет хранить состояние**

Том **форматируется** и **монтируется** в Pod/Container

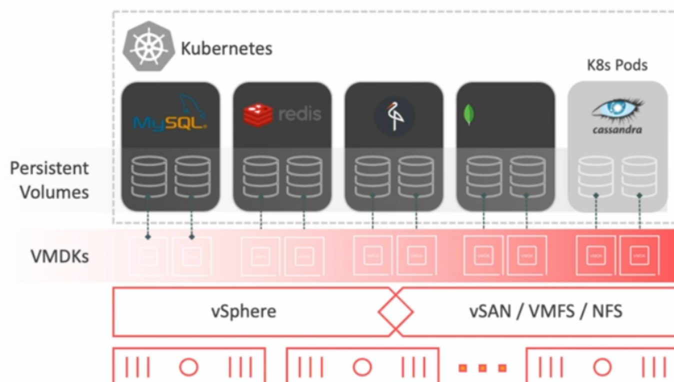
Рассмотрим ситуацию. Кластер Kubernetes развернут в формате ВМ поверх виртуальной инфраструктуры. Нам нужны Persistent Volumes, то есть какое-то устройство хранения данных, где будет храниться state. При этом жизненный цикл этого устройства не должен зависеть от жизненного цикла контейнера и ВМ, которая обеспечивает работу Container Host.

Для этого в VMware есть First Class диски (FCD, ранее — IVD). Если вы знакомы с инфраструктурой на VMware, вы знаете, что базовый элемент хранения данных — это VMDK диск. FCD похожи на VMDK, но есть одно отличие. Если VMDK привязан к виртуальной машине, то FCD — независимый объект со своим уникальным номером (UID), никак не связанный с жизненным циклом ВМ. Удаление ВМ не повлияет на FCD, и вы сможете переподключить его. Также один FCD можно подключить ко множеству ВМ (например, такой подход активно используется в VDI).

FCD создается на любом datastore. Обращение к данным в этом случае будет исключительно блочным. Если вы храните данные на vSAN, который является объектным хранилищем, FCD \при подключении внутрь ВМ будет блочным устройством, на которое происходит запись. Мы можем его монтировать, создавать, удалять независимо от того, как работает остальная инфраструктура.

Это происходит в два этапа. После создания нужно подключить FCD к виртуальной машине, на которой работает Container Host, а затем примонтировать в Pod/Container.

PersistentVolume (PV) — место хранения данных



Устройство, где храним актуальные данные

Жизненный цикл независим от контейнеров

Может быть создан статически или динамически

В динамическом режиме вы никогда не создаете эти устройства в ручную

Вернемся к Kubernetes

Базовый примитив хранения в Kubernetes — PersistentVolume (PV). Именно там Kubernetes хранит свои данные. Жизненный цикл PV не зависит от контейнера.

```
1. mylesgray@:~ (zsh)
~ cat PersistentVolumeClaim.yaml

kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: vsan-default
```

Ресурсы хранения запрашиваются пользователем:

PersistentVolume в виде статичного пути (Static Provisioning)

StorageClass (Dynamic Provisioning)

Процедура запроса PV называется PersistentVolumeClaim — в этом случае вы как бы «просите» подключить к контейнеру существующий или новый PV.

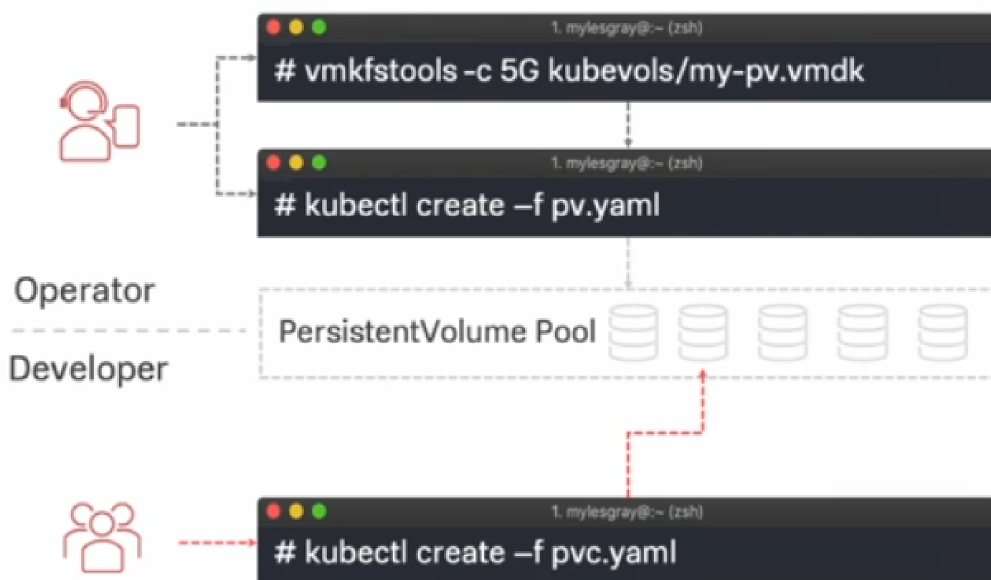
- PersistentVolume (PV) — устройство хранения
- PersistentVolumeClaim (PVC) — запрос на подключение/добавление тома к контейнеру

В запросе ресурсов описываются параметры и свойства PV, которые вам нужны.

Есть два способа запроса ресурсов: статический и динамический.

Static Provisioning — ручной режим

1. Ops создает каждый FCD/VMDK вручную.
2. Ops регистрирует VMDK в Kubernetes как PV, прописывая путь.
3. Разработчик назначает PV через PersistentVolumeClaim.
4. Разработчик создает Pod, который использует PVC.

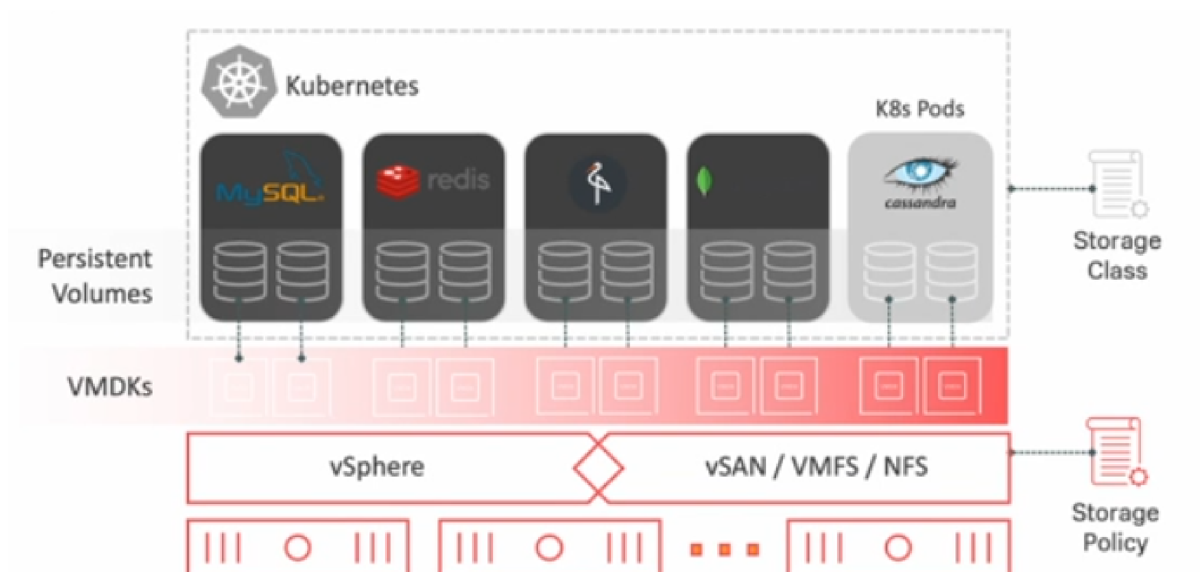


Недостатки этого подхода:

- большое количество ручных операций — нужно постоянно создавать и подключать тома;
- низкая управляемость при больших масштабах.

Для решения этих проблем был придуман StorageClass (SC) — способ динамически назначать устройства и управлять ими.

StorageClass позволяет описывать и передавать общие требования/характеристики без углубления в детали и отдавать их на уровне инфраструктуры. SC похож на VMware Storage Policy, который решал ту же задачу изоляции инфраструктуры от потребителя.



Так как StoragePolicy и StorageClass идентичны с точки зрения логики архитектуры, достаточно просто маппить напрямую SC в SP. Это и обеспечивает динамический provisioning и автоматизацию процессов.

При создании SC:

- даем название;
- описываем, с кем будет взаимодействие (provisioner);
- передаем параметры (в случае с VMware — имя StoragePolicy).

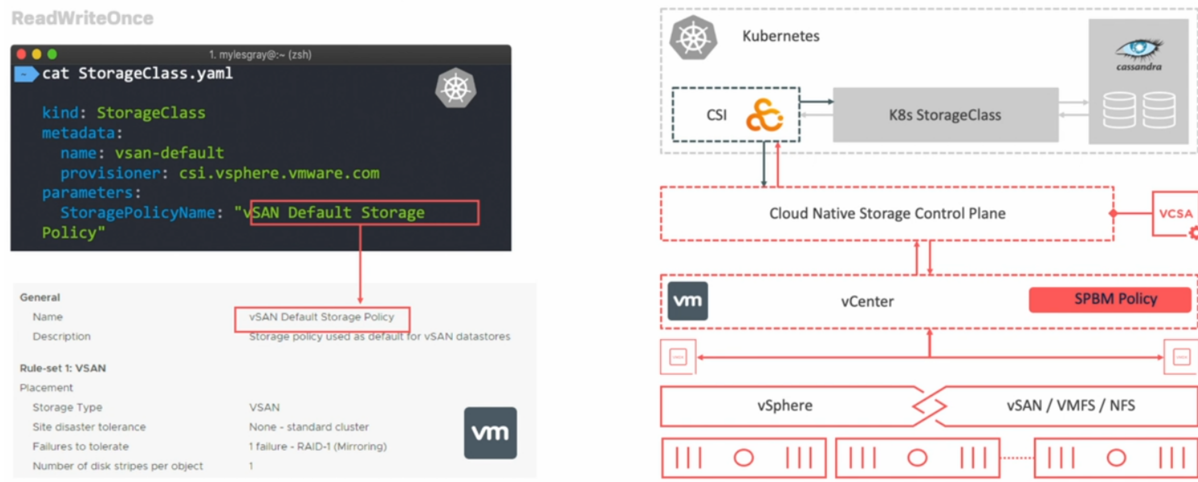
```
1. mylesgray@:~ (zsh)
~ cat StorageClass.yaml

  kind: StorageClass
  metadata:
    name: vsan-default
    provisioner: csi.vsphere.vmware.com
  parameters:
    StoragePolicyName: "vSAN Default Storage Policy"
```

Dynamic Provisioning позволяет упростить процесс развертывание устройств хранения:

1. Ops создает StorageClass.
2. Разработчик направляет PVC в StorageClass.
3. Система автоматически создает VMDK и PV.
4. Система подключает PV к PVC.

Пример. На схеме ниже разработчикам было предоставлено три типа хранилищ (Gold, Silver, Bronze). Для каждого из них можно указать собственные StoragePolicy. После предоставления необходимого набора ресурсов разработчик в запросе PVC указывает, какой SC ему нужен. Система автоматически создает FCD как PV и «отдает» его разработчику. Подключение и управление размещением происходят автоматически и не требуют вовлечения ни Ops, ни Dev.

Приблизительно так выглядит динамическое размещение блочного PV:

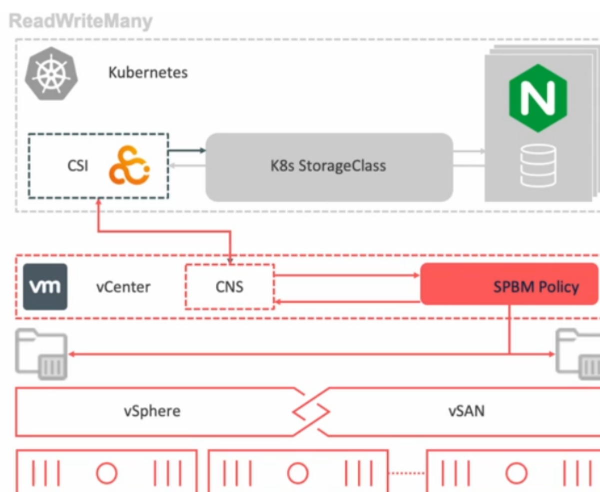


Блочное хранилище — это всегда **ReadWriteOnce**, Один контейнер имеет доступ к одним данным, так как в блочном хранилище в целях обеспечения консистентности данных всегда необходимо решать задачу изоляции.

Однако в некоторых случаях может понадобиться использование **ReadWriteMany** — чтобы один ресурс был подключен к многим подам для упрощения управления.

Пример. Ферма веб-серверов, которые работают с одним и тем же контентом.

Динамическое размещение файлового PV



Необходимо, когда **много контейнеров должны иметь доступ к одним** и тем же данным.

Реализуется по файловым, а не блочным протоколам (т.к. нужно управлять блокировками/доступами).

Необходимо иметь аналогичное динамическому развертывание.

Для реализации потребуется файловый доступ: у VMware это реализуется через динамическое создание файловых шар поверх vSAN.

> **Примечание:** это возможно, только если в качестве системы хранения используется vSAN — в его функционал входит динамическое создание шар по запросу и динамическое управление правами.

Единственное отличие: в StorageClass указывается другой тип доступа — файловый. В результате создается новая файловая шара, права на которую предоставляются в нужный нам контейнер.

```
1. mylesgray@:~ (zsh)
~ cat StorageClass.yaml

  kind: StorageClass
  apiVersion: storage.k8s.io/v1
  metadata:
    name: vsan-file
  provisioner: csi.vsphere.vmware.com
  parameters:
    storagepolicyname: "vSAN Default Storage Policy"
    csi.storage.k8s.io/fstype: nfs4
```

Динамически создает файловую шару на vSAN

ACL и RW указывается в момент создания

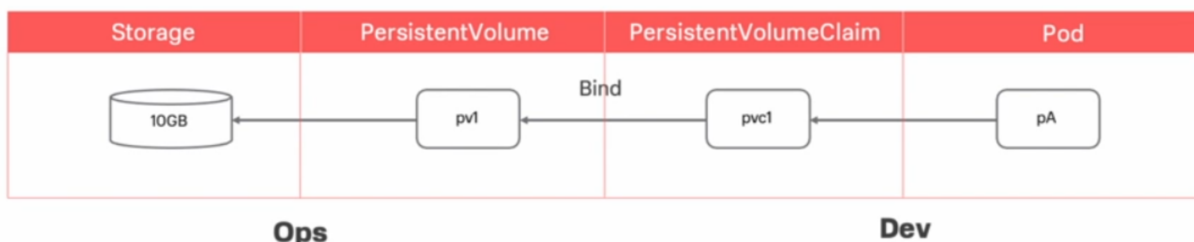
Требования к хранению через тот же SPBM, аналогично RWO

Полностью автоматизированный процесс управления шарами

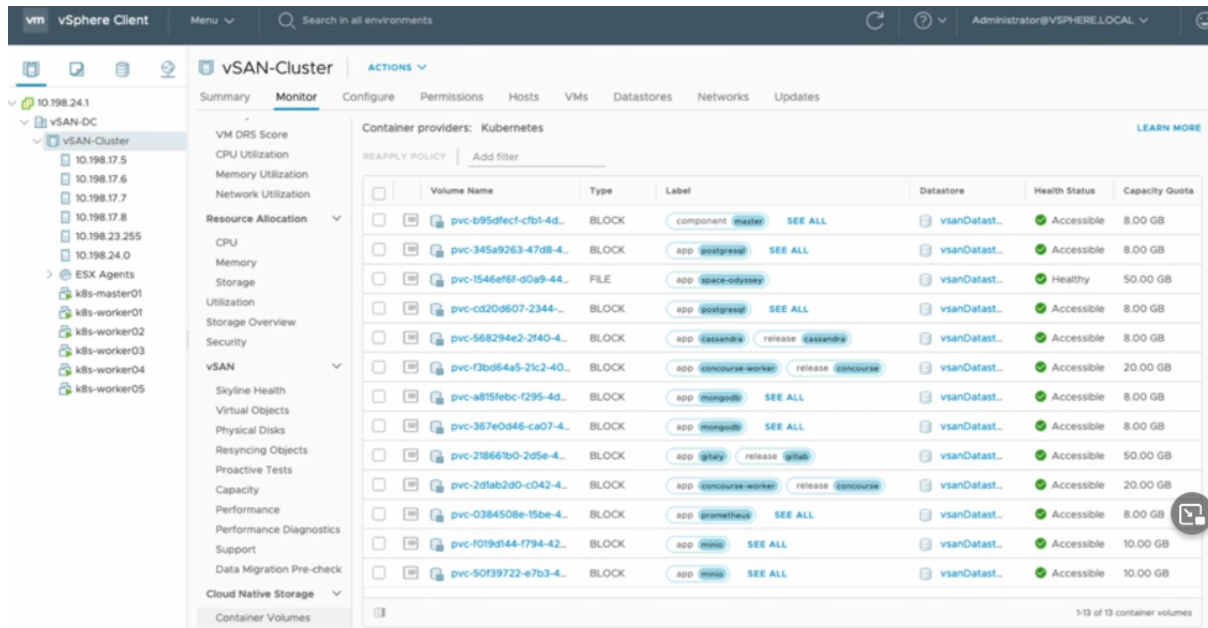
В итоге

С точки зрения инфраструктуры выглядит это следующим образом.

- У нас есть под, к которому мы подключаем устройство хранения.
- PersistentVolumeClaim «привязывает» PersistentVolume — некий диск на одном из хранилищ.
- Ops-команда отвечает за Storage и PersistentVolume.
- Dev-команда отвечает за Pods и запросы подключения (PVC).

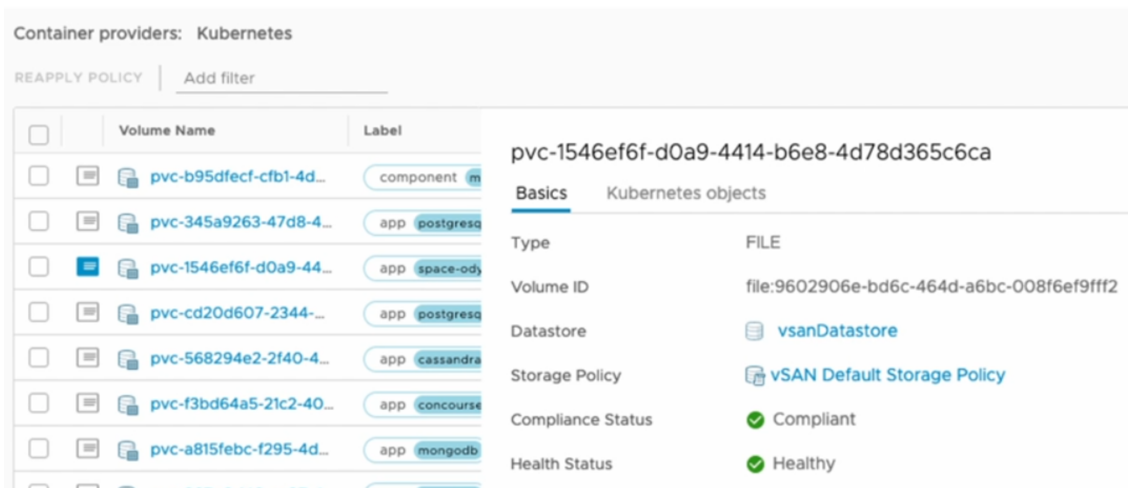


В консоли vSphere можно посмотреть все Volumes, созданные на этом кластере, их типы (блочные, файловые и т. д.), размер, состояние, место их хранения и метки Kubernetes.



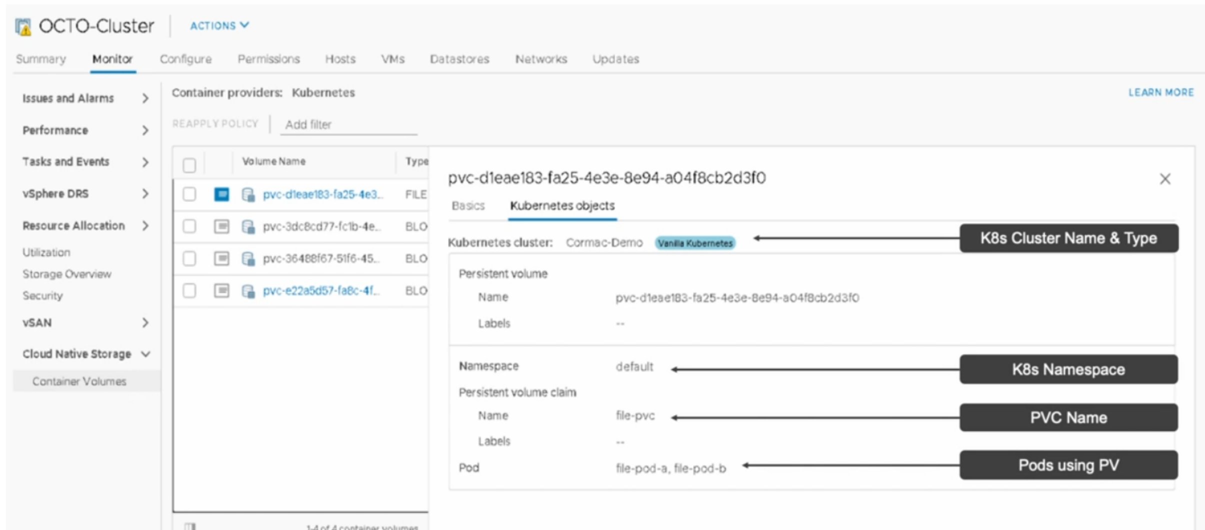
Зачем они нужны?
Позволяют быстро через фильтр найти PV, посмотреть его метрики доступности, производительности, где он находится и прочее.

Также с помощью Volume ID можно сопоставить Kubernetes Volumes с объектами vSphere.

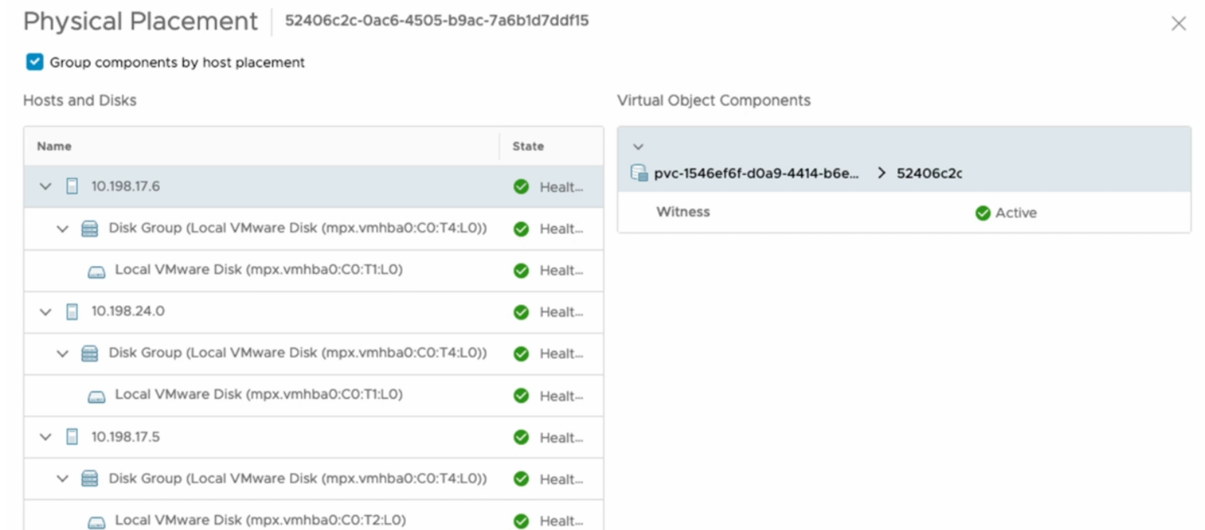


Можно посмотреть:

- имя и тип кластера;
- namespace;
- поды, которые используют PV.



Все это позволяет сделать хранение более прозрачным.



А также посмотреть, как система чувствует себя с точки зрения емкости и производительности.