

Модуль 4, урок 2



Вы уже знаете, из чего состоит кластер Kubernetes: из master-, worker-ноды и сервисов, которые на них запускаются. В контексте vSphere with Tanzu не происходит ничего нового.

Здесь тоже есть master- и worker-ноды, а также все необходимые процессы. Например, каждый хост виртуализации, каждый гипервизор — worker-нода одного большого кластера Kubernetes, который называется Supervisor Cluster. Используя этот кластер, в рамках демо мы запускали веб-сервер Nginx, который, будучи подом, запустился на worker-ноде, то есть на одном из ESXi-хостов.

На каждой worker-ноде работает сервис kubelet, который является прослойкой, позволяющей Kubernetes управлять движком контейнеризации. То есть на каждой worker-ноде работает движок контейнеризации, который выполняет всю работу по запуску контейнеров. Kubernetes не обращается к движку напрямую, он обращается к сервису kubelet. В vSphere with Tanzu этот сервис называется Spherelet.

Вспомните демо: в разделе Namespaces было три виртуальные машины, которые называются SupervisorControlPlaneVM. Таким длинным словом обозначаются master-ноды Supervisor Cluster, внутри которых запущены все необходимые служебные сервисы: kube-apiserver (сервер, который публикует все нужные нам эндпоинты API), база данных etcd (хранит конфигурацию кластера и всех приложений, которые запускаются внутри кластера) и kube-scheduler (отвечает за запуск подов на worker-нодах).

Является ли рабочая область, внутри которой мы запускали под с сервером Nginx, в том числе и Namespace, то есть классическим Namespace из мира Kubernetes, потому что в консоли управления это выглядело как Namespace?

Да, является. Если мы говорим о том, что внутри vSphere with Tanzu работает большой Supervisor Cluster, то, создавая Namespaces, которые мы намеренно называли рабочими проектами, мы в том числе создаем и Namespaces в кластере Kubernetes.

Внутри vSphere with Tanzu каждый такой Namespace имеет собственный ресурсный пул, который позволяет ИТ-администратору контролировать затраты на ресурсы, которые разработчик использует при запуске приложений внутри Namespace.

Какие типы нагрузок мы можем запускать внутри каждого Namespace?

Мы уже говорили про поды — нативные или vSphere-поды, которые максимально интегрированы в среду vSphere with Tanzu и представляют собой классические поды мира Kubernetes. Внутри пода крутится один или несколько контейнеров, внутри контейнеров — приложения. Под виден ИТ-администратору — он видит его утилизацию, может контролировать утилизацию, читать YAML-файлы и прочее.

Также в Namespace разработчик может запускать виртуальные машины. Но это немного не та ВМ, которую вы привыкли запускать на платформе VMware vSphere. Это виртуальная машина — объект в мире Kubernetes. Разработчик может взять YAML-файл с описанием этой ВМ и сказать Supervisor Cluster, что хочет запустить ее.

Все это работает на Custom Resources: VMware дополнила API Kubernetes в своем Supervisor Cluster так, чтобы разработчику были доступны объекты типа «виртуальная машина». Как любой другой объект, она описывается декларативно, и Kubernetes следит за тем, чтобы любое изменение между декларативным описанием ВМ и ее актуальным состоянием устранялось.

Зачем это нужно? Представим себе сценарий, где у нас есть большое монолитное приложение, которое разработчики планируют разбить на

микросервисы. Часть из этих микросервисов без проблем ложится в форм-фактор контейнеров, а вот с частью сервисов возникают проблемы. Разработчик понимает, что трудозатраты на перевод этих сервисов в форм-фактор контейнеров несоизмеримо большие и в этом нет необходимости или на это нет времени. При этом разработчик понимает, как такой сервис запустить в виде VM. Для реализации этого сценария и был придуман подход запуска VM как объектов Kubernetes.

Разработчик берет YAML-файл с описанием VM (см. слайд). Первой строчкой в качестве `kind` мы указали `virtual machine` — не под, не сервис, а именно виртуальную машину. После того как разработчик выполнит команду `kubectl create` и укажет YAML-файл, Kubernetes создаст объект типа «виртуальная машина».

Эту VM необходимо как-то кастомизировать. Для этого в продукте используется open-source-проект под названием `cloud-init`. Он позволяет запускать специальные скрипты, которые выполняются при первом старте VM, — с их помощью можно кастомизировать систему нужным образом. Например, задать создание SSH-ключей, чтобы получить доступ к VM по SSH, создать пользователя с определенными правами или установить конкретный пакет из конкретного репозитория. `Cloud-init` предлагает практически безграничные возможности по кастомизации вашей операционной системы.

После того как разработчик описал спецификацию VM на языке YAML, провел необходимые настройки с помощью `cloud-init`, он может получить доступ к этой VM так же, как он делает это с обычным подом. Для этого он связывает объект типа `Service` с объектом типа «виртуальная машина», используя лейблы и селекторы. Например, если нам нужно опубликовать доступ по SSH до VM, мы создаем специальный сервис.

Если нам нужен доступ только внутри кластера — это будет сервис типа `Cluster API`, если нужен доступ извне — сервис с типом `LoadBalancer`. Логика работы та же, что и для подов. Платформа публикует внешний IP-адрес, если мы используем `LoadBalancer`. Обращаясь на этот IP-адрес, разработчик получает доступ к своей VM — как и с подами.

Третий тип объектов, который разработчик может запускать внутри `Namespace`, — вложенный кластер. Это набор VM, внутри которых запущено и настроено все необходимое, чтобы этот набор функционировал как «ванильный» кластер Kubernetes без каких-либо изменений. Внутри кластера разработчик может запускать приложения, изменять API, писать свои `Custom Resources` и так далее. Все то же самое, что он может делать с абсолютно ванильным `Upstream Kubernetes`.

Работает это на основе open-source-проекта Cluster API. Его идея заключается в том, чтобы запускать кластеры Kubernetes как приложения, используя один большой Management Cluster Kubernetes. В vSphere with Tanzu им выступает Supervisor Cluster. Тот самый, у которого каждый ESXi-хост, каждый хост гипервизора — это worker-нода.

Итак, кластер Kubernetes запускается как приложение внутри кластера Kubernetes. Значит, это приложение можно описать на языке разметки YAML. На слайде вы видите пример описания вложенного кластера. Это достаточно простое описание, которое подразумевает три master-ноды, три worker-ноды и версию кластера Kubernetes, соответствующую версии 1.20.7.

В чем удобство такого подхода? Он позволяет обеспечивать разработчику режим самообслуживания. Тот самый, который мы упоминали в рамках разговора про концепцию запуска Kubernetes в продуктиве. Например, у разработчика есть приложение, которое прекрасно чувствует себя в среде Kubernetes версии 1.18. Ему нужно протестировать приложение в среде Kubernetes версии 1.20. Разработчик может за несколько минут, используя готовый шаблон, запустить внутри своего Namespace кластер Kubernetes версии 1.20, а затем начать тестировать приложение.

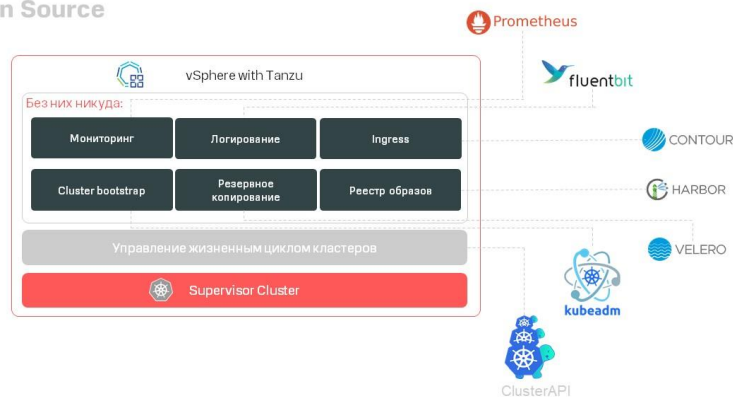
Используя декларативное описание, можно пойти дальше и встраивать процесс создания кластера Kubernetes в цикл CI/CD с помощью инструментов автоматизации — например, Terraform или vRealize Operation. [Они доступны в облаке #CloudMTS.](#)

Все, что от вас требуется, — просто оформить необходимый запрос к API Supervisor Cluster. Это справедливо не только для установки и создания кластера Kubernetes, но и для любой операции по управлению жизненным циклом кластеров. Например, если вы хотите обновить версию кластера Kubernetes с 1.18 до 1.20. Для этого нужно выполнить команду `kubectl patch` или `kubectl edit` и выбрать нужный кластер или изменить YAML-файл и выполнить команду `apply`, используя уже измененный файл. В ответ на это вы получите классический RollingUpdate кластера, за это будет ответственна платформа VMware vSphere.

Для масштабирования кластера — то же самое. Используя команду `patch` или изменяя YAML-файл, вы переприменяете декларативное описание, увеличиваете количество нод, и vSphere with Tanzu ответственно за то, чтобы привести это декларативное описание в единство с текущим состоянием. Если вы хотите вместо трех worker-нод иметь пять, vSphere with Tanzu постепенно добавит еще две worker-ноды в ваш кластер.

vSphere with Tanzu

и дружба с Open Source



Тут представлены приложения, снижавшие популярность в мире cloud-native-разработки:

- Prometheus — приложение для сбора метрик как с уровня инфраструктуры Kubernetes, так и с уровня приложений, запущенных внутри кластера.
- Fluent Bit — делает то же самое для логов приложений.
- Contour — выполняет роль ингресс-контроллера.

Все эти open-source-проекты встроены в vSphere with Tanzu.

Конечно, их можно установить нативно: взять готовый Helm Chart под приложение или YAML-спецификацию установки. Однако, если вы оперируете множеством кластеров Kubernetes и часто используете комбинацию из разных приложений, гораздо проще использовать один унифицированный подход по управлению жизненным циклом приложений. Например, разработчику удобнее всего было бы выполнять команду вида `application install`, подставляя нужное приложение в конце. Платформа должна сама понять, что необходимо установить, откуда и как.

Такой подход реализован в рамках фреймворка Extensions, который позволяет унифицировано управлять жизненным циклом популярных приложений из мира Cloud Native. Для разработчика это упрощает управление и использование таких приложений. Для администратора это будут понятные компоненты решения, и он точно будет знать, что они провалидированы вендором и проверены на совместимость с платформой.

Говоря про архитектуру платформы запуска сред Kubernetes, нельзя обойти стороной вопрос сети. Сам Kubernetes не ответственен за то, как будет настроена сеть для ваших приложений, подов и сервисов, которые вы запускаете внутри кластера. Kubernetes лишь отдает спецификацию, Container Network Interface. Каждый вендор и каждый разработчик может написать свой сетевой плагин, который необходимо установить внутри

кластера Kubernetes. Этот плагин и будет отвечать за выделение IP-адресов подам, сервисам, обеспечивать гарантированную доставку обращений от подов к сервисам или, наоборот, гарантированно не доставлять обращения, если мы говорим про безопасность Kubernetes и используем Kubernetes Network Policy.

Не будем углубляться в детали и технические реализации сетевого стека для vSphere with Tanzu. Нужно лишь понимать, что платформа запуска сред Kubernetes в продуктиве из коробки должна обеспечивать весь необходимый сетевой стек. vSphere with Tanzu это успешно делает.

Такая же история с Container Storage Interface. Kubernetes не ответственен за реализацию логики работы stateful-приложений с системой хранения данных. За это отвечает Storage Plugin через спецификацию, которая определяет Kubernetes.

Бесплатная редакция платформы — Tanzu Community Edition

Редакция использует все те же подходы и компоненты, что и коммерческая версия Tanzu. Нет никаких лимитов на использование. Вы можете скачать дистрибутив платформы и запустить ее:

- на своей рабочей станции, используя движок контейнеризации Docker;
- на одной из трех облачных платформ — vSphere, AWS, MS Azure.

Open-source-компонентов в Tanzu Community Edition даже больше, потому что первые экспериментальные пакеты появляются именно в бесплатной редакции.

Тем, кто интересуется вопросами безопасности кластера Kubernetes, наверняка знакомы проекты вроде Open Policy Agent или Gatekeeper. Например, в Tanzu Community Edition, выполняя команду вида `application install gatekeeper`, можно получить автоматизированную установку Open Policy Agent вместе с фреймворком Gatekeeper и начать писать политики безопасности, узнавать, как это работает.

Создание кластеров Kubernetes и запуск приложений

Итак, мы разработчики. Нам нужно запустить вложенный кластер Kubernetes и начать в нем работу — создать внутри какое-то приложение.

Как мы уже выяснили, кластеры Kubernetes запускаются с помощью описания на языке YAML. Давайте посмотрим на пример такого описания.

```
apiVersion: run.tanzu.vmware.com/v1alpha1
kind: TanzuKubernetesCluster
metadata:
  name: demo-cluster
  namespace: mtscloud
spec:
  topology:
    controlPlane:
      count: 1
      class: best-effort-small # vnclass to be used for master(s)
      storageClass: gold
    workers:
      count: 1
      class: best-effort-small # vnclass to be used for workers(s)
      storageClass: gold
      # volumes:
      #   - name: containerd
      #     mountPath: /var/lib/containerd
      #     capacity:
      #       storage: 5Gi
  distribution:
    version: v1.20.7
  settings:
    network:
      cni:
        name: antrea
      services:
        cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
```

1. Отправим кластер на создание командой `kubectl apply -f`, указывая путь до YAML-спецификации. Система оповестит нас, что кластер создан.

```
roma@ubuntu:~$ vim apps/cluster20.yaml
roma@ubuntu:~$ kubectl apply -f apps/cluster20.yaml
tanzukubernetescluster.run.tanzu.vmware.com/demo-cluster created
roma@ubuntu:~$ k
```

2. Проверяем, в каком мы контексте, командой `current-context`. Мы должны быть в контексте `mtsccloud`.

```
roma@ubuntu:~$ kubectl config current-context
mtsccloud
```

3. Выполним команду `kubectl get tkc` внутри этого контекста. Видим, что у нас есть один демокластер, создание которого мы только что запустили. Сейчас он в стадии создания.

```
roma@ubuntu:~$ kubectl get tkc
NAME                CONTROL PLANE  WORKER  DISTRIBUTION                AGE   PHASE   TKR CO
MPATIBLE  UPDATES AVAILABLE
demo-cluster      1                1      v1.20.7+vmware.1-tkg.1.7fb9067  31s  creating  True
```

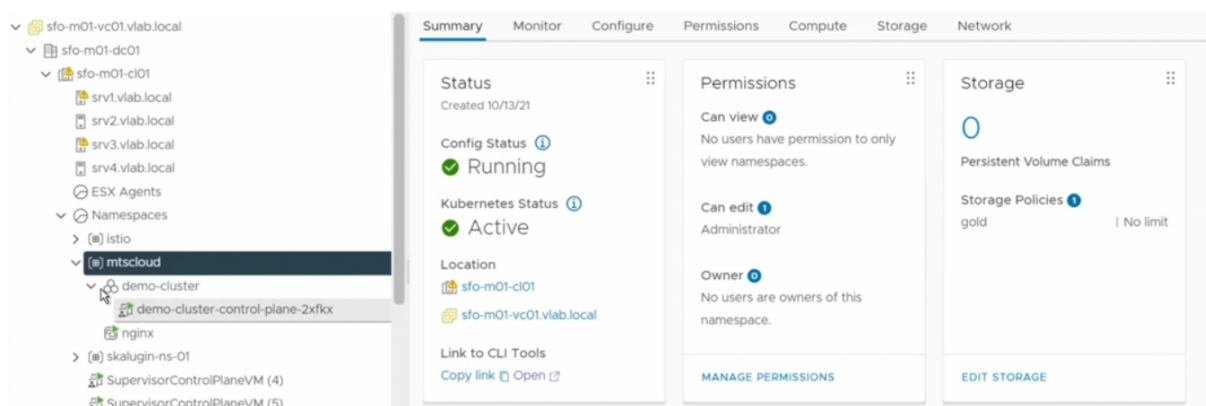
4. Вернемся к спецификации и посмотрим, что она из себя представляет.

```
apiVersion: run.tanzu.vmware.com/v1alpha1
kind: TanzuKubernetesCluster
metadata:
  name: demo-cluster
  namespace: mtsccloud
spec:
  topology:
    controlPlane:
      count: 1
      class: best-effort-small # vmclass to be used for master(s)
      storageClass: gold
    workers:
      count: 1
      class: best-effort-small # vmclass to be used for workers(s)
      storageClass: gold
      volumes:
      - name: containerd
        mountPath: /var/lib/containerd
        capacity:
        storage: SGI
  distribution:
    version: v1.20.7
  settings:
    network:
      cni:
        name: antrea
    services:
      cidrBlocks: ["198.51.100.0/12"]
    pods:
      cidrBlocks: ["192.0.2.0/16"]
```

- Первые две строчки — эндпоинт API-кластера Kubernetes, к которому мы обращаемся.
- Вторая говорит, что тип объекта, который мы просим Kubernetes создать, — это `TanzuKubernetesCluster` (кастомный ресурс, которым расширен API Supervisor Cluster Kubernetes).

- Namespace соответствует названию Namespace, в котором мы работаем.
- Спецификация вложенного кластера:
 - одна master- и одна worker-нода;
 - класс master- и worker-нод будет соответствовать классу best-effort-small (определяет количество процессора и оперативной памяти, которые будут выделяться VM, формирующим вложенный кластер).
- storageClass определяет политику хранения, которую мы, будучи ИТ-администраторами, сделали доступной внутри Namespace. Stateful-приложениям, которые разработчик захочет запускать внутри этого кластера, будет выделяться место в хранилище в соответствии с этой политикой хранения.
- Версия кластера Kubernetes — 1.20.7.

С точки зрения ИТ-администратора, мы можем увидеть, что внутри Namespace mtscloud был создан такой объект под названием demo-cluster и в нем уже запущена виртуальная машина.



Конкретно эта VM — master-нода. Когда произойдет процесс запуска и конфигурирования, пройдут все нужные healthcheck'и, система продолжит создавать кластер и запустит worker-ноду.

Управление жизненным циклом кластеров

Перейдем в Namespace istio и посмотрим на cluster888.

```
roma@ubuntu:~$ kubectl config use-context istio
Switched to context "istio".
roma@ubuntu:~$ █
```

Если выполнить команду `get tks` внутри этого контекста, мы получим список из двух кластеров Kubernetes.

```
roma@ubuntu:~$ kubectl get tks
NAME          CONTROL PLANE  WORKER  DISTRIBUTION  AGE  PHASE  TKR COMP
ATIBLE  UPDATES AVAILABLE
cluster888    1              2      v1.20.7+vmware.1-tkg.1.7fb9067  88d  running  True
cluster999    1              3      v1.20.7+vmware.1-tkg.1.7fb9067  88d  updating  True
roma@ubuntu:~$ █
```

Предположим, что в cluster888 нам требуются не две worker-ноды, как это сейчас, а три. Откроем на редактирование спецификацию кластера.

```
roma@ubuntu:~$ kubectl edit tks cluster888 █
```

Найдем в спецификации раздел, где указано количество worker-нод, и изменим 2 на 3.

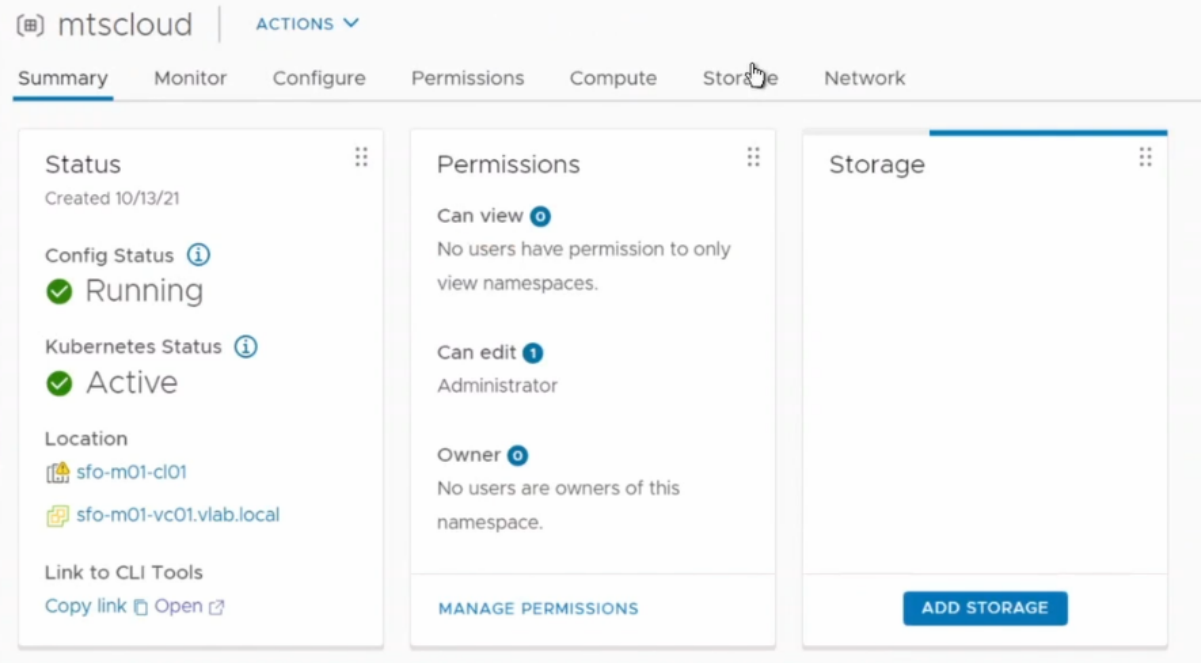
```
topology:
  controlPlane:
    class: best-effort-medium
    count: 1
    storageClass: gold
  workers:
    class: best-effort-large
    count: 3 █
    storageClass: gold
```

Сохраним файл. После этого vSphere with Tanzu добавит worker-ноду в кластер.

Изменение версии дистрибутива тоже делается через команду edit. Находим версию, меняем значение и сохраняем файл. vSphere with Tanzu сделает классический Rolling Update нашего кластера Kubernetes.

```
spec:
  distribution:
    fullVersion: v1.20.7+vmware.1-tkg.1.7fb9067
    version: v1.21
  settings:
    network:
      cni:
        name: antrea
```

Посмотрим, как себя чувствует наш вложенный кластер из Namespace mtscloud. Он находится в статусе Running.



The screenshot shows the vSphere with Tanzu console interface for the namespace 'mtscloud'. The 'Summary' tab is active, displaying the following information:

- Status:** Created 10/13/21
- Config Status:** Running (indicated by a green checkmark)
- Kubernetes Status:** Active (indicated by a green checkmark)
- Location:** sfo-m01-cl01 and sfo-m01-vc01.vlab.local
- Link to CLI Tools:** Copy link and Open

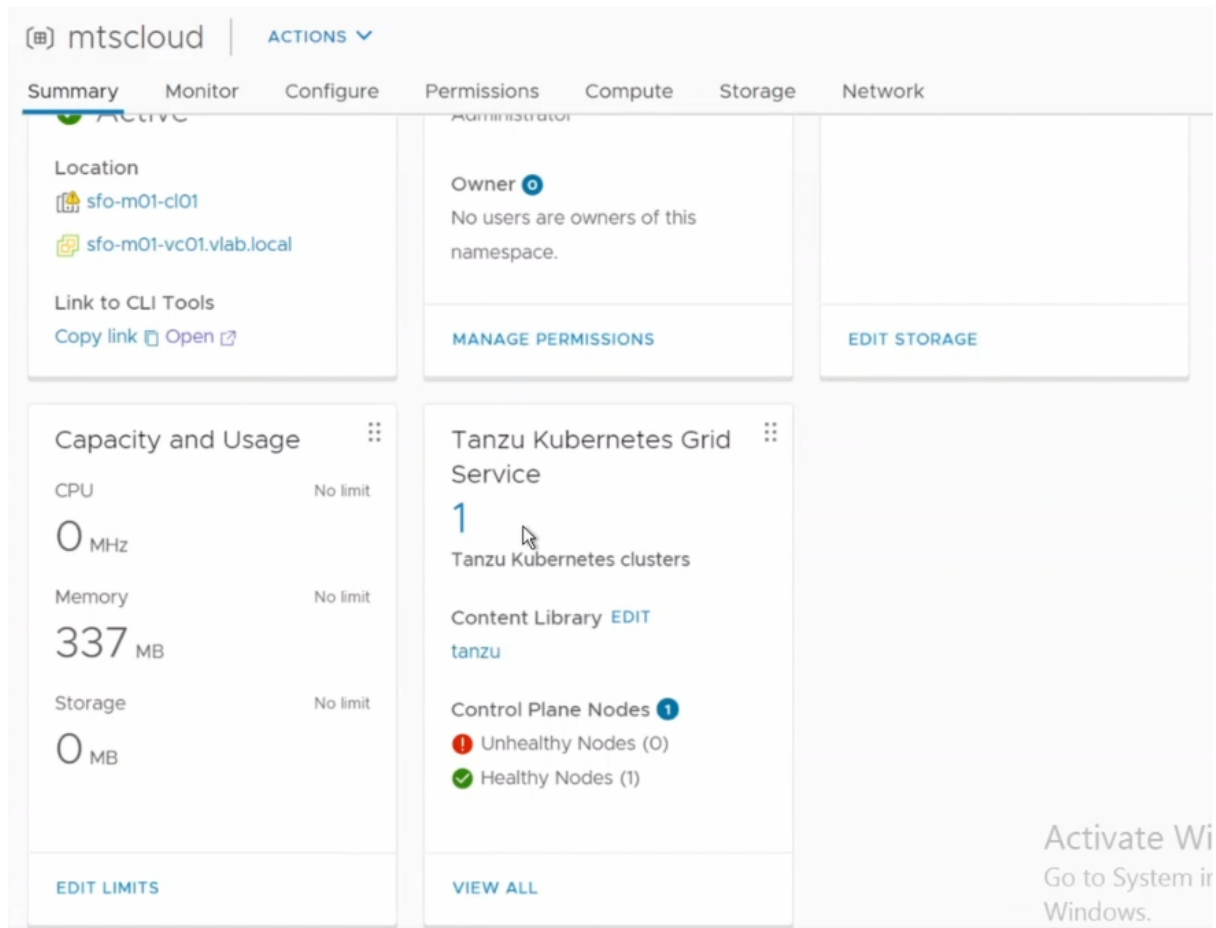
The 'Permissions' tab shows:

- Can view:** No users have permission to only view namespaces.
- Can edit:** Administrator
- Owner:** No users are owners of this namespace.
- MANAGE PERMISSIONS** button

The 'Storage' tab is currently empty, with an **ADD STORAGE** button at the bottom.

Администратор инфраструктуры может смотреть за количеством и статусом вложенных кластеров Kubernetes из консоли управления vSphere Client.

На вкладке Tanzu Kubernetes Grid Service есть цифра 1. Она указывает количество развернутых внутри Namespace кластеров.



Будучи разработчиком, мы хотим перейти внутрь этого кластера и запустить там приложение. Что нужно сделать?

Выполним команду `login` и укажем конкретный Namespace, в котором живет вложенный кластер `mtscloud`, и имя кластера, в который мы хотим пройти авторизацию, — `demo-cluster`.

```
roma@ubuntu:~$ kubectl vsphere login --server 192.168.124.1 --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify --tanzu-kubernetes-cluster-namespace mtscloud --tanzu-kubernetes-cluster-name demo-cluster
```

Используемая учетная запись соответствует той учетной записи, права на которую ИТ-администратор выдал в рамках настройки Namespace.

Авторизация успешно пройдена. Меняем контекст — он будет соответствовать имени кластера. Теперь разработчик может выполнять стандартные команды Kubernetes — например, попросить список нод.

```
roma@ubuntu:~$ kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
demo-cluster-control-plane-2xfkx   Ready    control-plane,master   4m43s   v1.20.7+vm
ware.1
demo-cluster-workers-wblh8-78974c9d99-7xw4z   Ready    <none>    2m40s   v1.20.7+vm
ware.1
```

Мы видим две ноды: одну master- и одну worker-ноду. Все соответствует тому, что мы видим из консоли управления vSphere Client.

Чтобы запустить приложение, нужно использовать все те же команды, которые разработчик привык выполнять, работая с обычным кластером Kubernetes. Например, мы хотим запустить веб-сервер Apache.

Создадим сервис типа LoadBalancer, чтобы мы могли обратиться к этому веб-серверу как внешний пользователь.

```
roma@ubuntu:~$ kubectl run httpd --image httpd
pod/httpd created
roma@ubuntu:~$ kubectl expose pod httpd --port 80 --type LoadBalancer
service/httpd exposed
```

Проверим статус пода, статус сервиса. Система выделила внешний IP-адрес, куда мы можем зайти через веб-браузер и увидеть, что веб-сервер Apache поднят и работает.

```
roma@ubuntu:~$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
httpd     1/1     Running   0           23s
roma@ubuntu:~$ kubectl get svc
NAME                TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
httpd               LoadBalancer  198.59.0.73  192.168.124.8 80:31530/TCP     6s
kubernetes          ClusterIP      198.48.0.1   <none>        443/TCP          5m36s
supervisor          ClusterIP      None         <none>        6443/TCP         5m31s
```

Так нехитро выглядит запуск кластера Kubernetes глазами разработчика. Этот подход дает невероятную гибкость: один разработчик может иметь множество кластеров Kubernetes, автоматизировать процесс их запуска с помощью соответствующих инструментов.

Вам нужно только сформировать шаблон кластера и сделать запрос к API Supervisor Cluster Kubernetes.