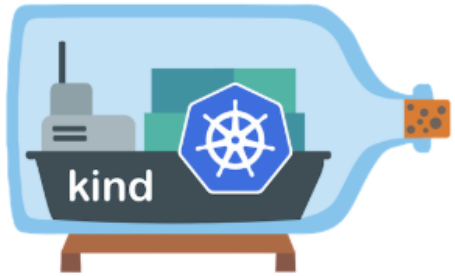
Linux Administration: Kubernetes

Main concepts

Kubernetes

About the platform

Kubernetes: Getting started



kubernetes

MicroK8s

minikube

What Is Kubernetes?

- According to the Kubernetes website,
- *"Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications."*
- **Kubernetes** comes from the Greek word **κυβερνήτης**, which means *helmsman* or *ship pilot*. With this analogy in mind, we can think of Kubernetes as the pilot on a ship of containers.
- Kubernetes is also referred to as **k8s** (pronounced Kate's), as there are 8 characters between *k* and *s*.

Kubernetes: Concepts

- To work with Kubernetes, you use Kubernetes API objects to describe your cluster's desired state:
  - what applications or other workloads you want to run,
  - what container images they use, the number of replicas,
  - what network and disk resources you want to make available, and more.

- You set your desired state by creating objects using the Kubernetes API, typically via the command-line interface, kubectl.
- You can also use the Kubernetes API directly to interact with the cluster and set or modify your desired state.

Kubernetes: Cloud platform

- Google Kubernetes Engine (GKE): Google's solution that manages production-ready Kubernetes clusters for you.
- Amazon Elastic Kubernetes Service (EKS): Amazon's solution that manages production-ready Kubernetes clusters for you.
- Azure Kubernetes Service (AKS): Azure's solution that provides you managed, production-ready Kubernetes clusters.
- OpenShift Kubernetes: Red Hat's solution that handles Kubernetes clusters for you.

Kubernetes: Features

Kubernetes: Features

- Kubernetes offers a very rich set of features for container orchestration. Some of its fully supported features are:
  - **Automatic bin packing**
    Kubernetes automatically schedules containers based on resource needs and constraints, to maximize utilization without sacrificing availability.
  - **Self-healing**
    Kubernetes automatically replaces and reschedules containers from failed nodes. It kills and restarts containers unresponsive to health checks, based on existing rules/policy. It also prevents traffic from being routed to unresponsive containers.
  - **Horizontal scaling**
    With Kubernetes applications are scaled manually or automatically based on CPU or custom metrics utilization.

Kubernetes: Features

- Kubernetes offers a very rich set of features for container orchestration. Some of its fully supported features are:
  - **Service discovery and Load balancing**
    Containers receive their own IP addresses from Kubernetes, while it assigns a single Domain Name System (DNS) name to a set of containers to aid in load-balancing requests across the containers of the set.
  - **Automated rollouts and rollbacks**
    Kubernetes seamlessly rolls out and rolls back application updates and configuration changes, constantly monitoring the application's health to prevent any downtime.
  - **Secret and configuration management**
    Kubernetes manages sensitive data and configuration details for an application separately from the container image, in order to avoid a re-build of the respective image. Secrets consist of sensitive/confidential information passed to the application without revealing the sensitive content to the stack configuration, like on GitHub.

Kubernetes: Features

- Kubernetes offers a very rich set of features for container orchestration. Some of its fully supported features are:
  - **Storage orchestration**
    Kubernetes automatically mounts software-defined storage (SDS) solutions to containers from local storage, external cloud providers, distributed storage, or network storage systems.
  - **Batch execution**
    Kubernetes supports batch execution, long-running jobs, and replaces failed containers.

Why Use Kubernetes?

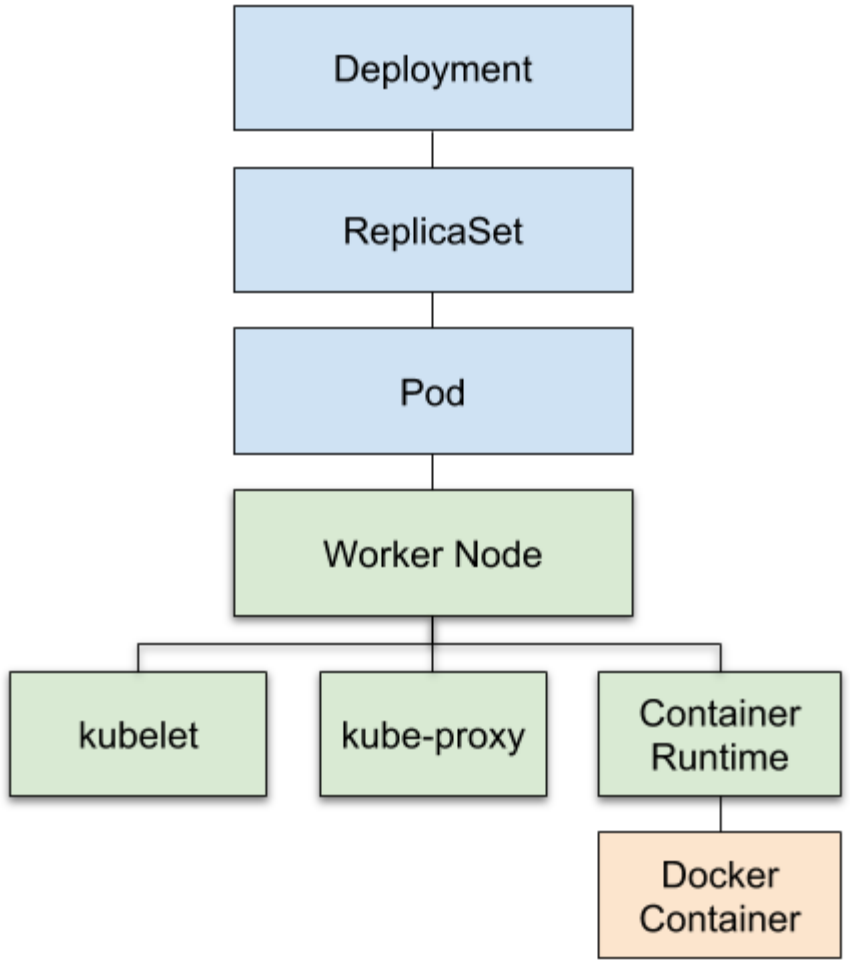
- In addition to its fully-supported features, Kubernetes is also portable and extensible. It can be deployed in many environments such as local or remote Virtual Machines, bare metal, or in public/private/hybrid/multi-cloud setups. It supports and it is supported by many 3rd party open source tools which enhance Kubernetes' capabilities and provide a feature-rich experience to its users.
- Kubernetes' architecture is modular and pluggable. Not only that it orchestrates modular, decoupled microservices type applications, but also its architecture follows decoupled microservices patterns. Kubernetes' functionality can be extended by writing custom resources, operators, custom APIs, scheduling rules or plugins.

Kubernetes

Main concepts

Kubernetes: Abstraction Level

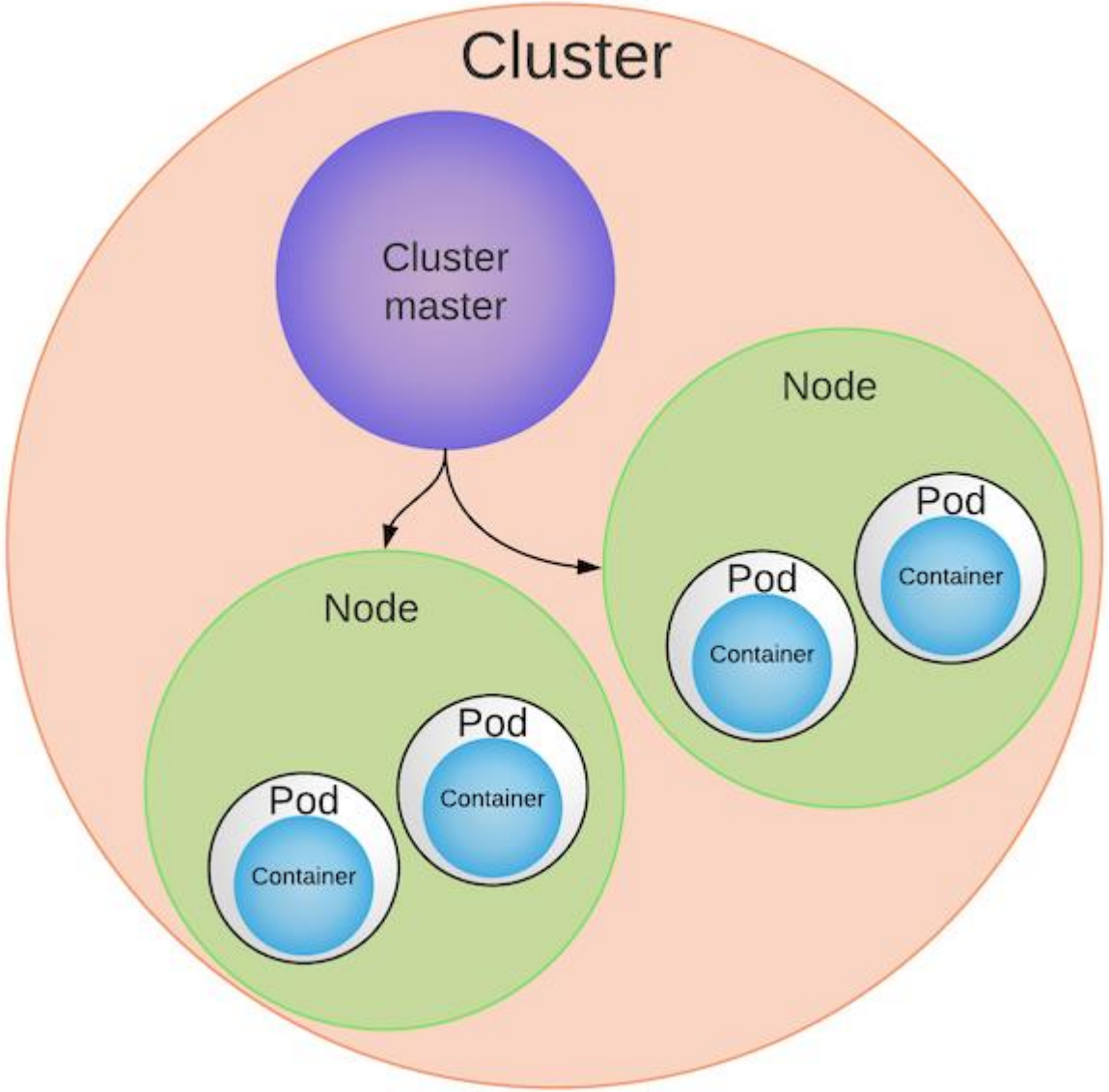
- Deployments create and manage ReplicaSets,
- which create and manage Pods, which run on Nodes,
- which have a container runtime,
- which run the app code you put in your Docker image.



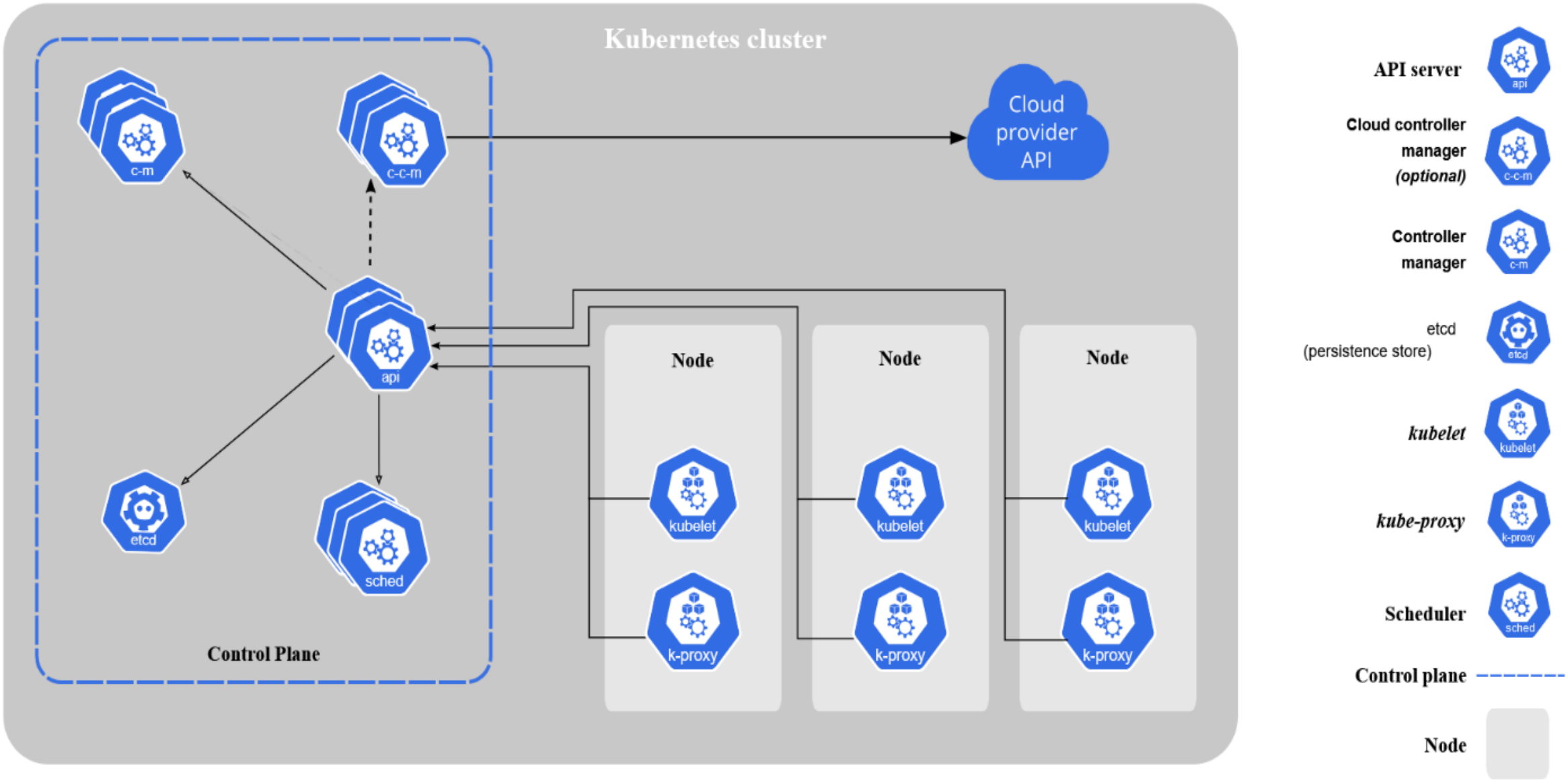
https://towardsdatascience.com/key-kubernetes-concepts-62939f4bc08e

Kubernetes cluster

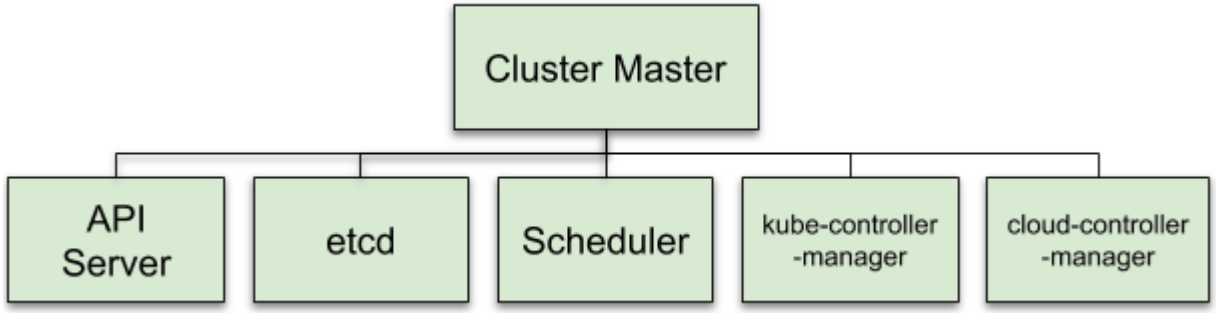
- **Masters** make scheduling decisions, respond to events, implement changes, and monitor the Cluster.
- **Worker node is** a computer server.
- One or more **Pods** run on a single **Worker Node**.



Kubernetes components

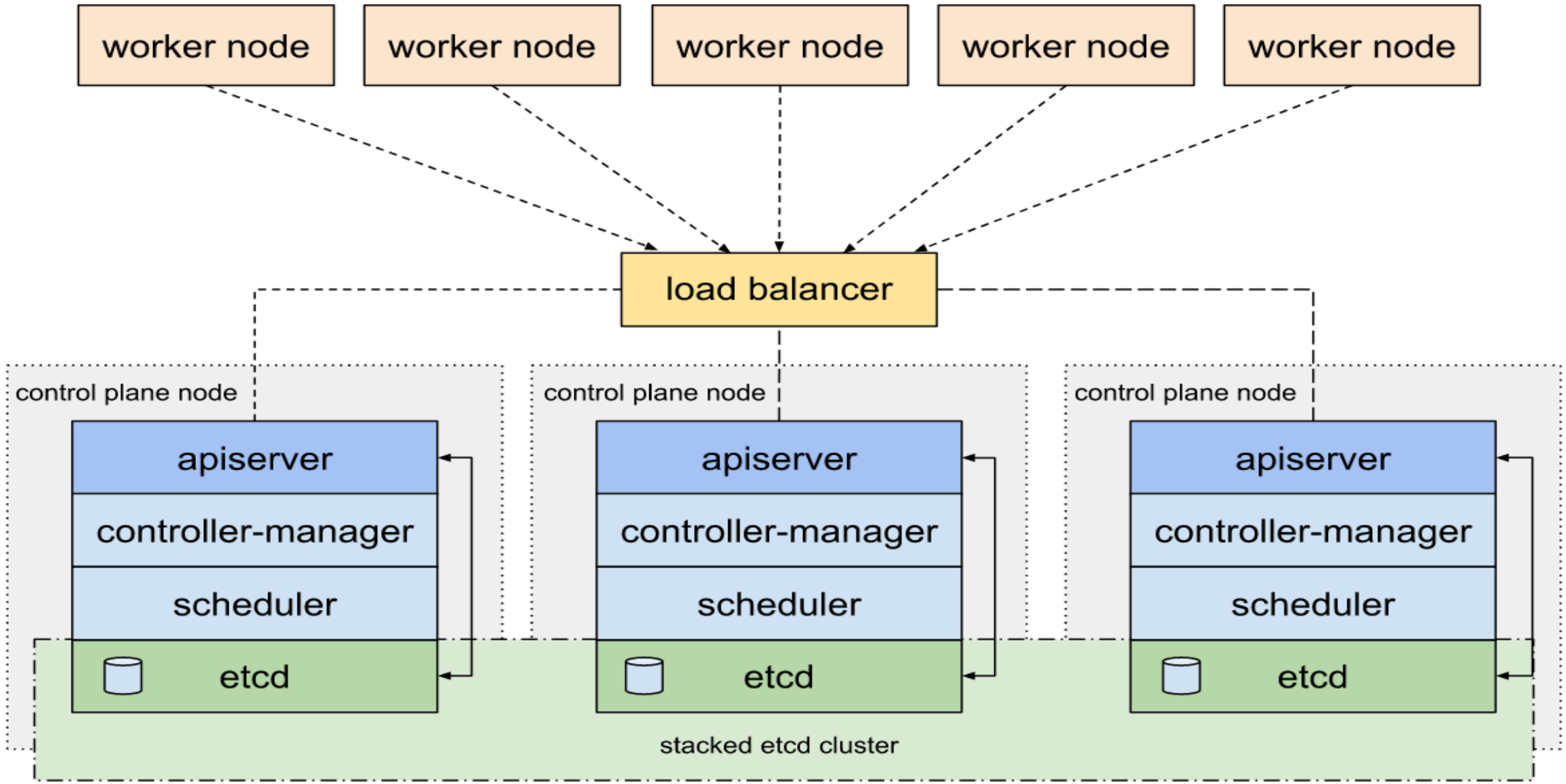


Kubernetes master

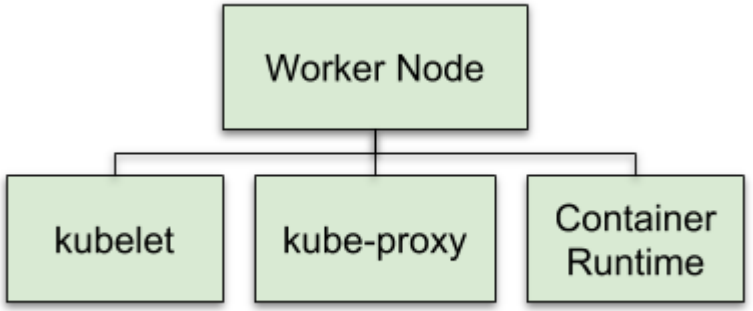


- **API Server** — Exposes the K8s API. It's the frontend for Kubernetes control. (aka. kube-apiserver) Think hub.
- **etcd** — Distributed key-value store for Cluster state data. Think Cluster info.
- **Scheduler** — Selects the Nodes for new Pods. Good guide here. (aka kube-scheduler) Think matcher.
- **kube-controller-manager** —Process that runs controllers to handle Cluster background tasks. Think Cluster controller.
- **cloud-controller-manager** — Runs controllers that interact with cloud providers. Think cloud interface.

kubeadm HA topology - stacked etcd



Kubernetes worker

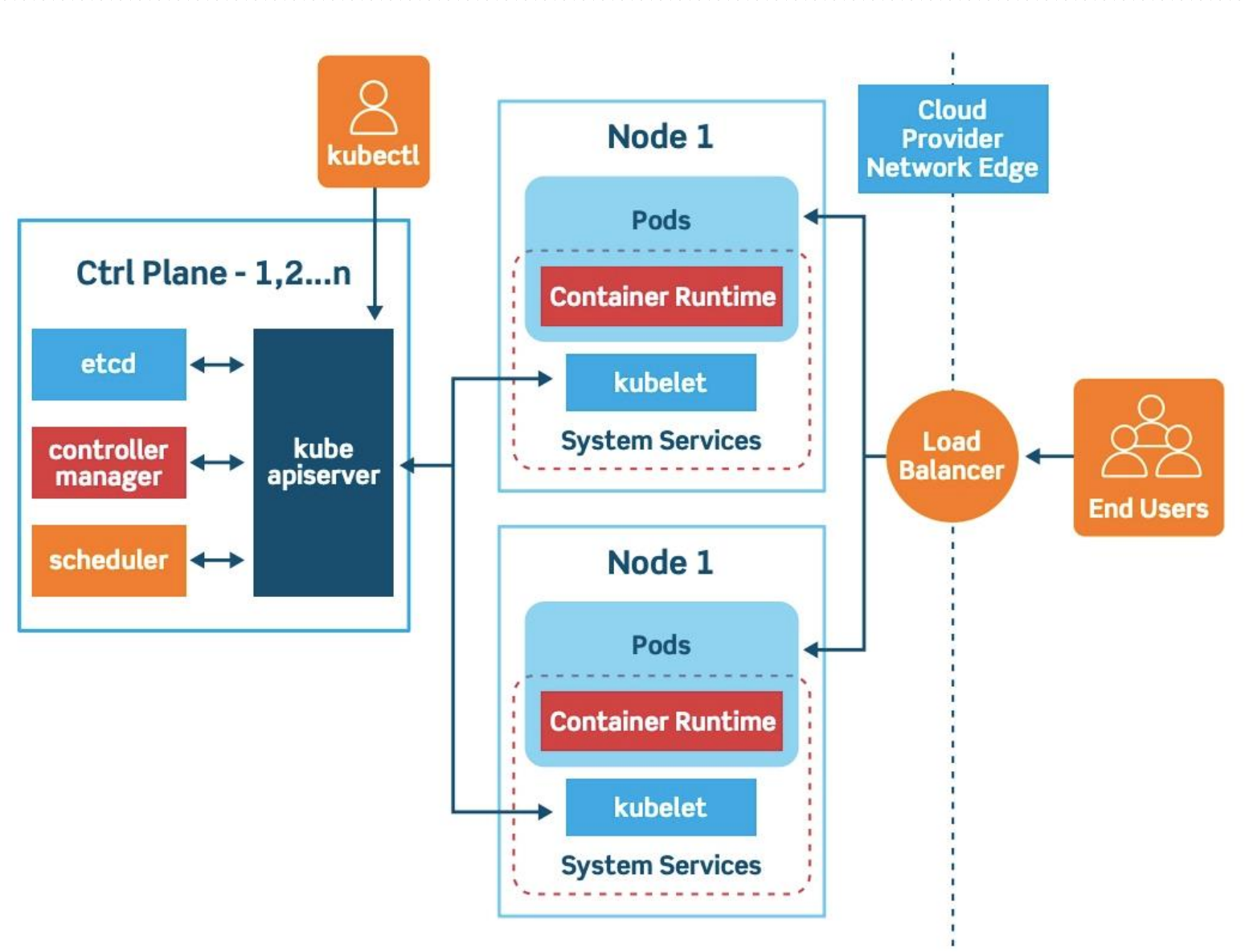


- **kubelet** — Responsible for everything on the Worker Node. It communicates with the Master's API server. Think brain for Worker Node.
- **kube-proxy** — Routes connections to the correct Pods. Also performs load balancing across Pods for a Service. Think traffic cop.
- **Container Runtime** — Downloads images and runs containers. For example, Docker is a Container Runtime. Think Docker.

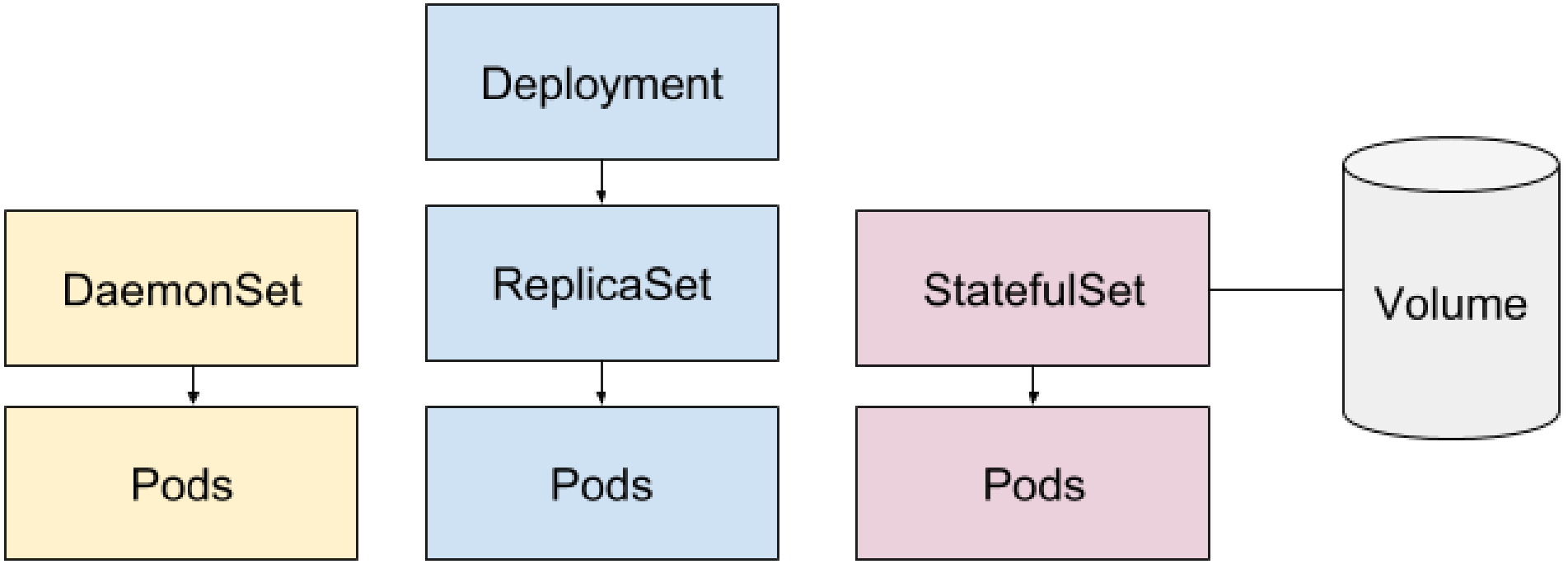
Kubernetes:

Functional Level:

- control plane
- pods



Kubernetes High-Level Controllers



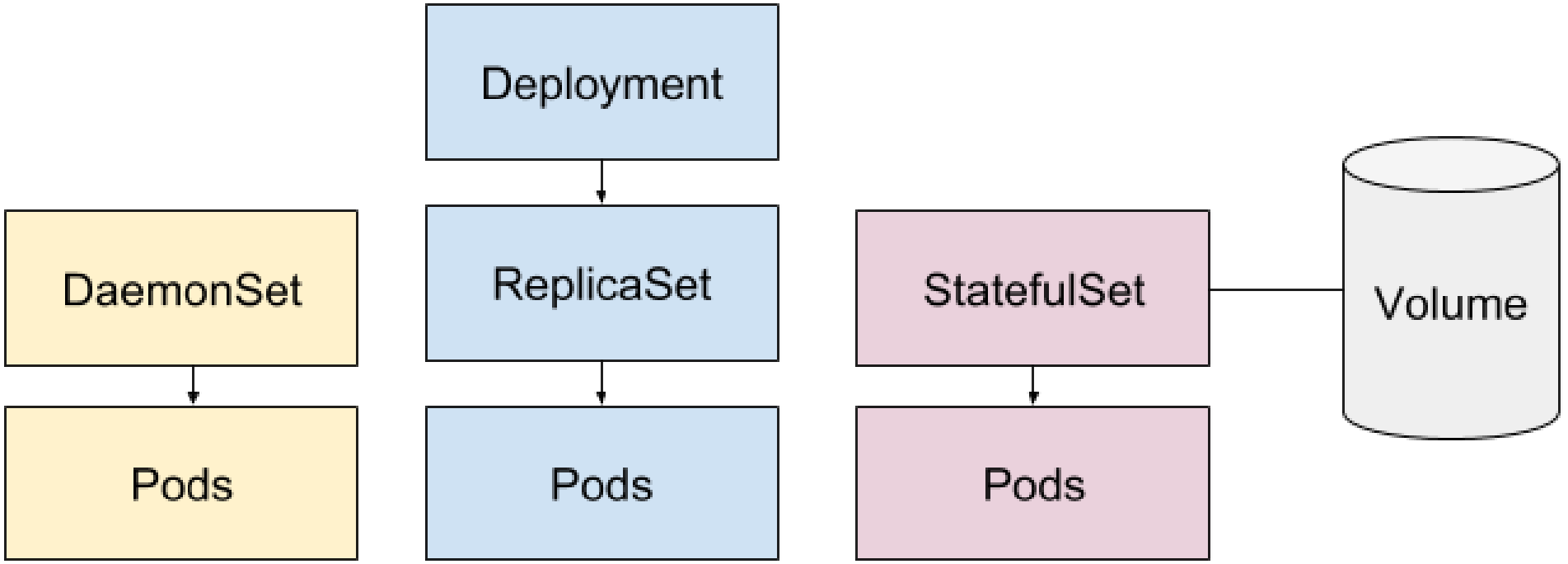
Kubernetes

Main concepts

Kubernetes Object Model

- Kubernetes has a very rich object model, representing different persistent entities in the Kubernetes cluster. Those entities describe:
  - What containerized applications we are running
  - The nodes where the containerized applications are deployed
  - Application resource consumption
  - Policies attached to applications, like restart/upgrade policies, fault tolerance, etc.
- Examples of Kubernetes objects are Pods, ReplicaSets, Deployments, Namespaces, etc. We will explore them next.

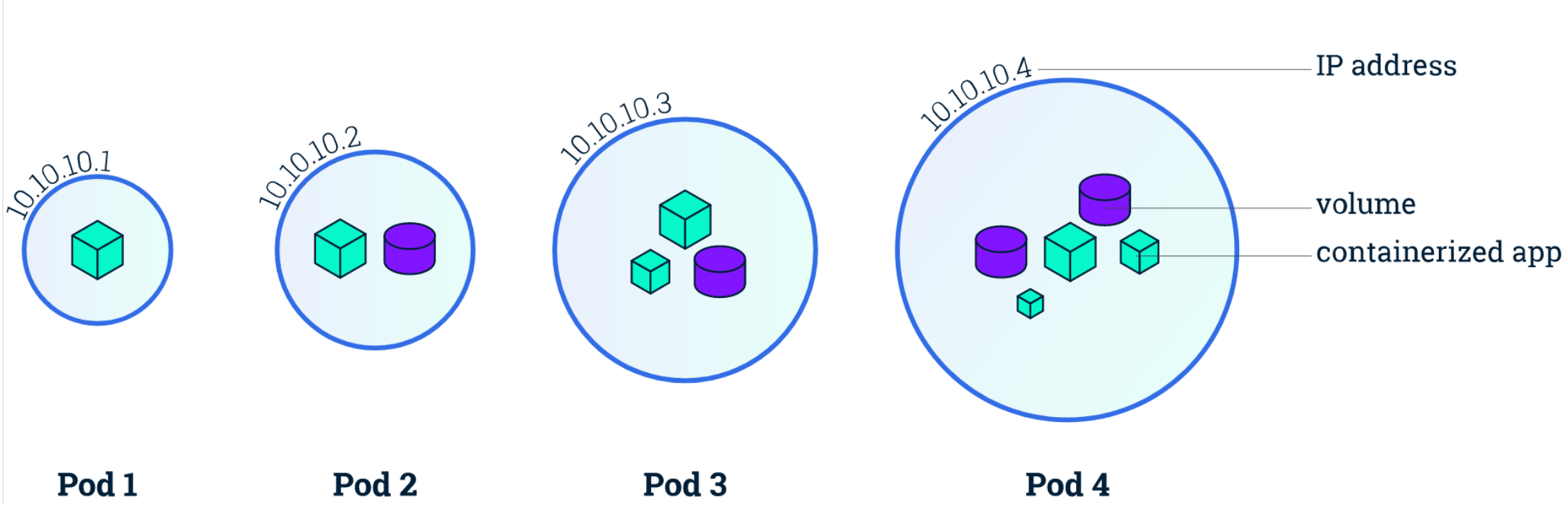
Kubernetes High-Level Controllers



Kubernetes Pods

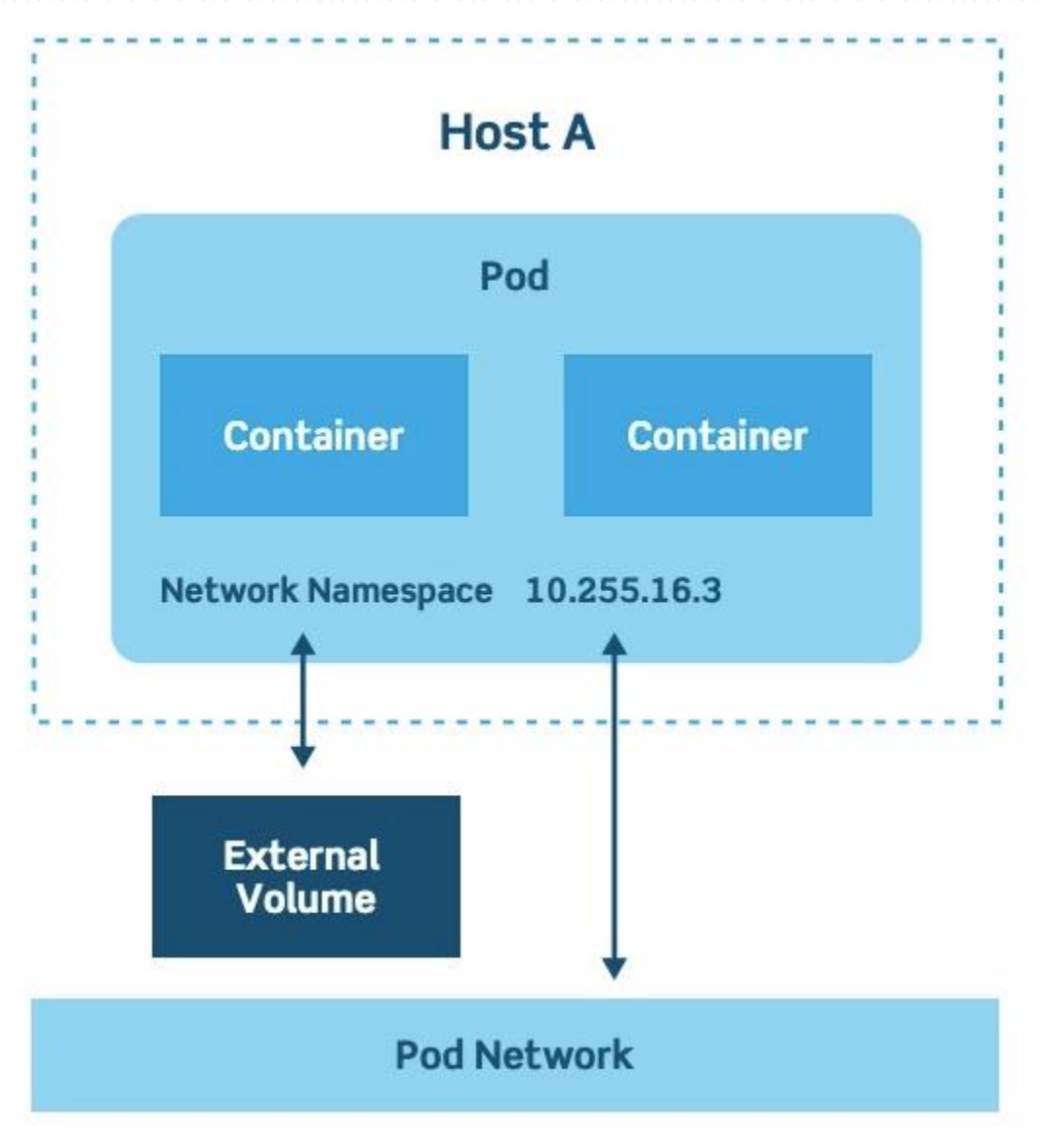
- Usually you don't need to create Pods directly, even singleton Pods. Instead, create them using workload resources such as Deployment or Job. If your Pods need to track state, consider the StatefulSet resource.
- Pods in a Kubernetes cluster are used in two main ways:
  - The "one-container-per-Pod" model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container;
  - A Pod can encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share resources. These co-located containers form a single cohesive unit of service—for example, one container serving data stored in a shared volume to the public, while a separate *sidecar* container refreshes or updates those files.

Kubernetes Pods



Kubernetes Pod

- **Pod**
- The Pod is the basic building block of Kubernetes. A Pod contains a group of one or more containers. Generally, each Pod has one container



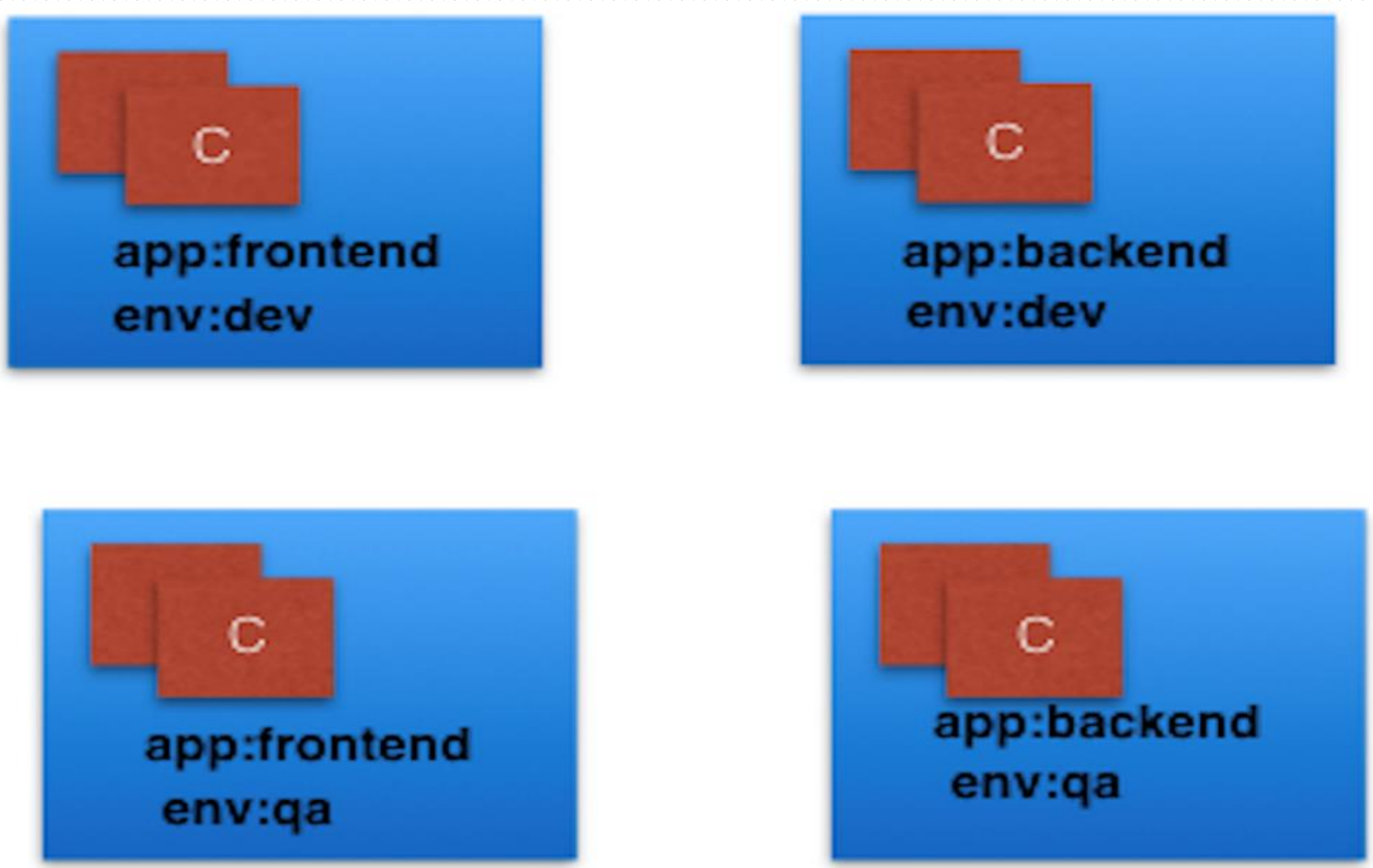
Kubernetes Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.15.11
    ports:
    - containerPort: 80
```

Labels

- **Labels** are **key-value pairs** attached to Kubernetes objects (e.g. Pods, ReplicaSets, Nodes, Namespaces, Persistent Volumes).
- Labels are used to organize and select a subset of objects, based on the requirements in place.
- Many objects can have the same Label(s).
- Labels do not provide uniqueness to objects.
- Controllers use Labels to logically group together decoupled objects, rather than using objects' names or IDs.

Labels


C
app:frontend
env:dev
C
app:backend
env:dev
C
app:frontend
env:qa
C
app:backend
env:qa

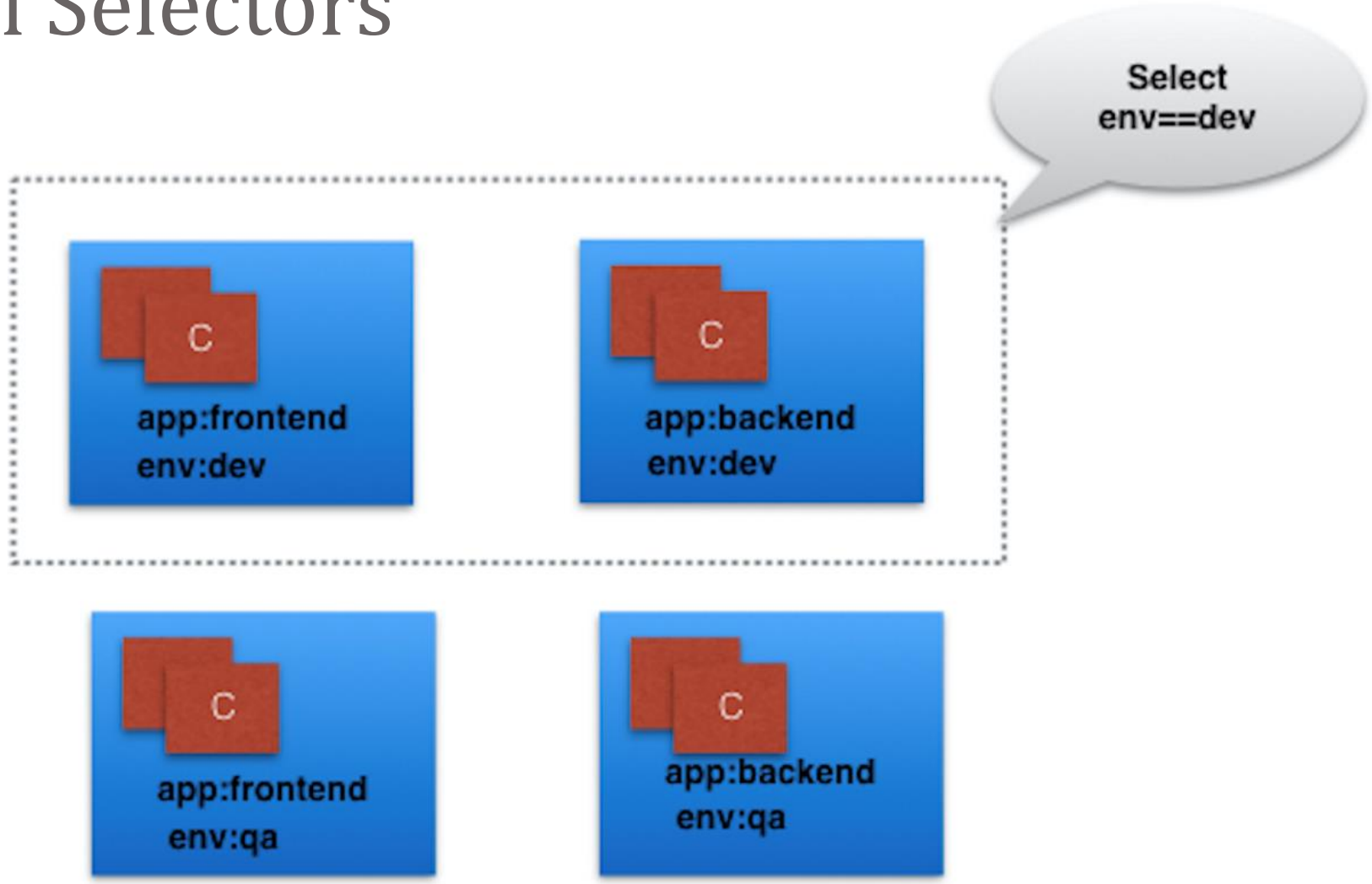
Kubernetes **Label Selectors**

- o Equality-Based Selectors
- Equality-Based Selectors allow filtering of objects based on Label keys and values. Matching is achieved using the =, == (equals, used interchangeably), or != (not equals) operators. For example, with env==dev or env=dev we are selecting the objects where the env Label key is set to value dev.
- o Set-Based Selectors
- Set-Based Selectors allow filtering of objects based on a set of values. We can use in, notin operators for Label values, and exist/does not exist operators for Label keys. For example, with env in (dev,qa) we are selecting objects where the env Label is set to either dev or qa; with !app we select objects with no Label key app.

Kubernetes Label

- In the image above, we have used two Label keys: **app** and **env**.
- Based on our requirements, we have given different values to our four Pods.
- The Label **env=dev** logically selects and groups the top two Pods, while the Label **app=frontend** logically selects and groups the left two Pods.
- We can select one of the four Pods - bottom left, by selecting two Labels: **app=frontend AND env=qa**.

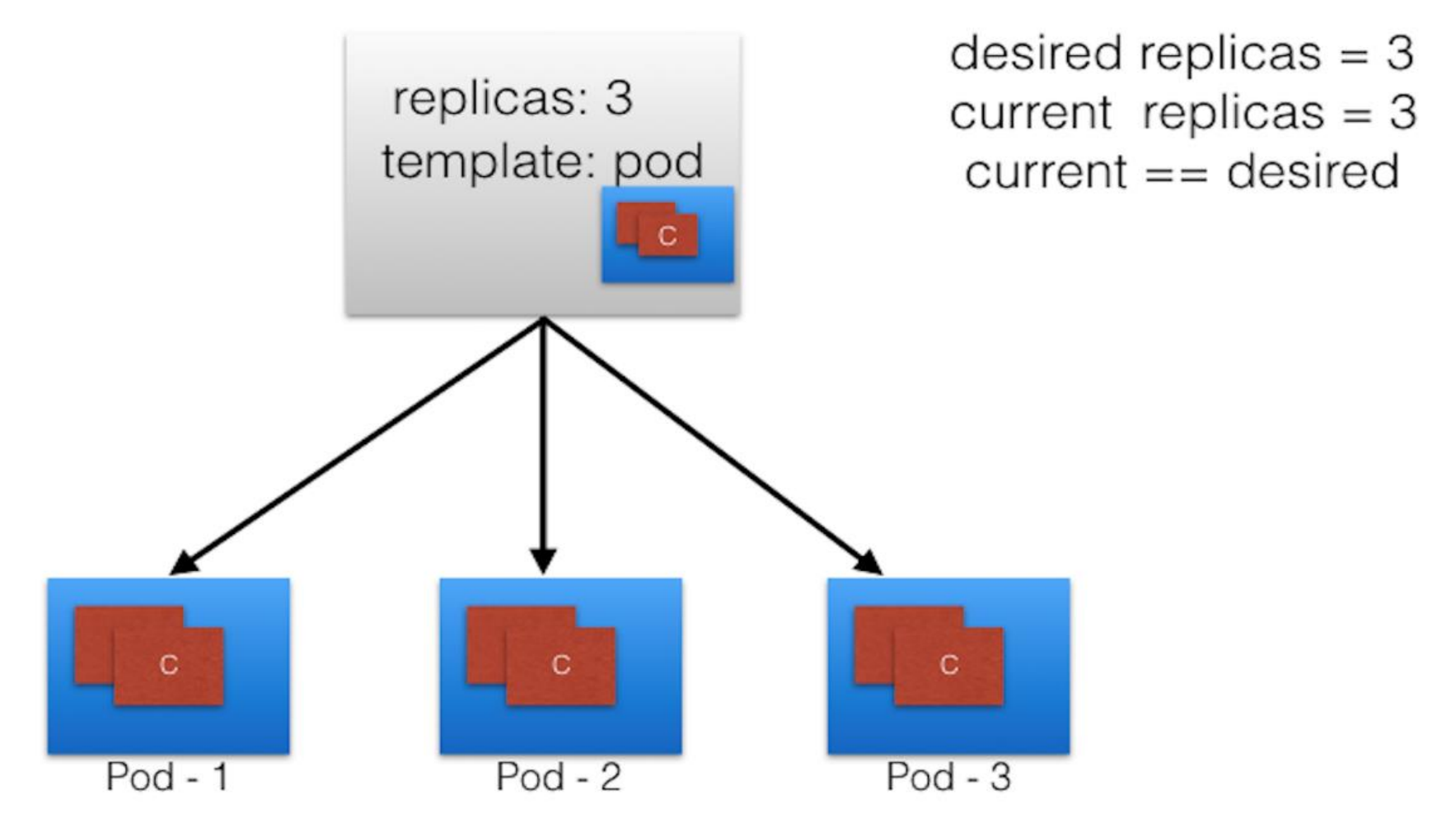
Label Selectors



Kubernetes ReplicaSets

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.
- A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria.

Replica Sets



Kubernetes ReplicaSets

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Kubernetes ReplicaSets Autoscaling

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Kubernetes **Namespaces**

- If multiple users and teams use the same Kubernetes cluster we can partition the cluster into virtual sub-clusters using **Namespaces**. The names of the resources/objects created inside a Namespace are unique, but not across Namespaces in the cluster.
- To list all the Namespaces, we can run the following command:
- kubectl get namespaces

```
NAME              STATUS      AGE
default           Active      11h
kube-node-lease   Active      11h
kube-public       Active      11h
kube-system       Active      11h
```

Kubernetes High-Level Controllers

- **StatefulSet**

- Like a ReplicaSet, a StatefulSet manages deployment and scaling of a group of Pods based on a container spec.
- Unlike a Deployment, a StatefulSet's Pods are not interchangeable. Each Pod has a unique, persistent identifier that the controller maintains over any rescheduling.
- StatefulSets for good for persistent, stateful backends like databases.

Kubernetes StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
```

Kubernetes High-Level Controllers

- DaemonSets are for continuous process.
- They run one Pod per Node.
- DaemonSets are useful for ongoing background tasks such as monitoring and log collection.
- Each new Node added to the cluster automatically gets a Pod started by the DaemonSet.
- As nodes are removed from the cluster, those Pods are garbage collected.
- Deleting a DaemonSet will clean up the Pods it created

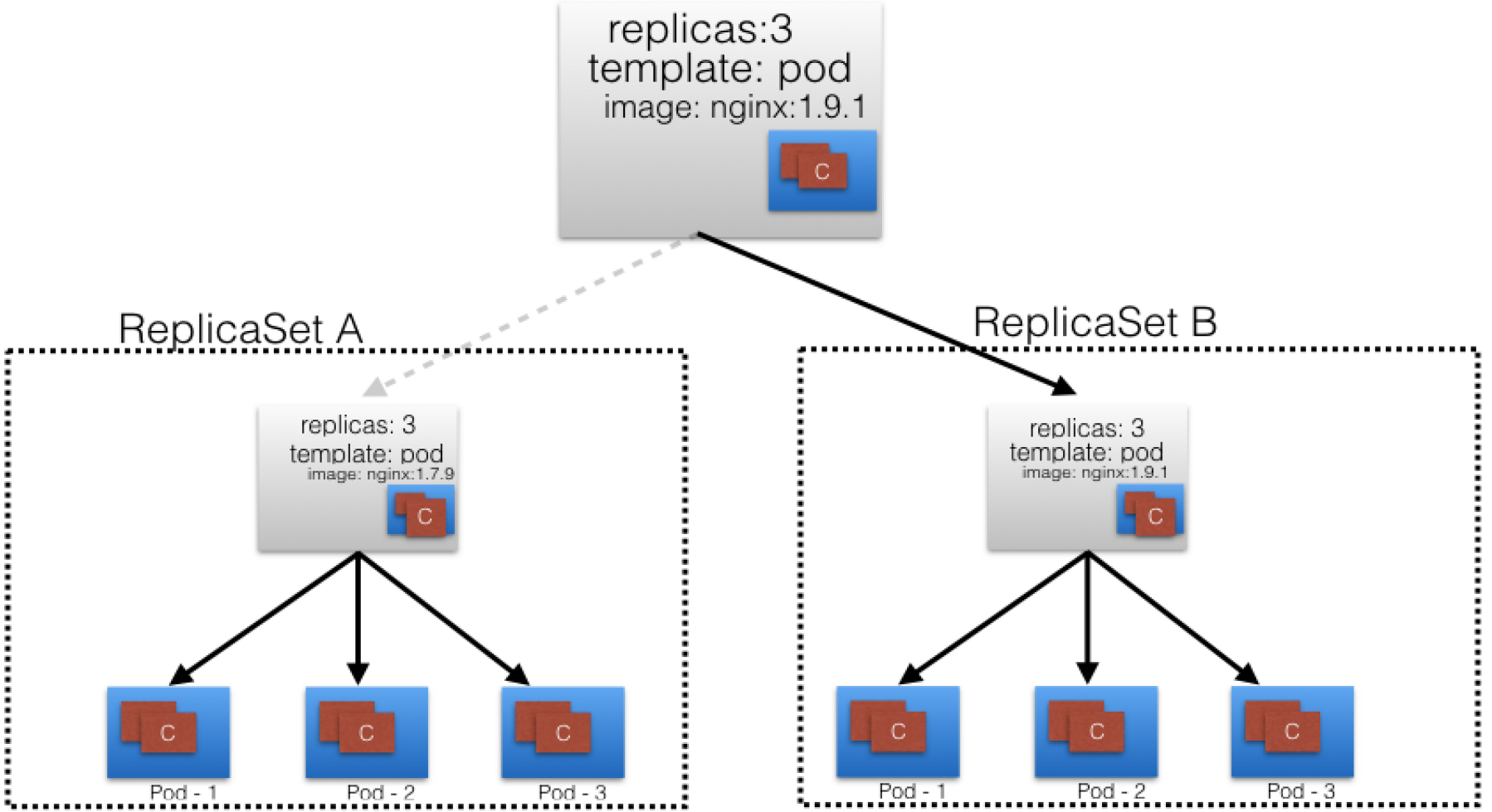
Kubernetes DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
.......
```

Kubernetes Deployments

- A Deployment provides declarative updates for Pods and ReplicaSets.
- You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Deployments



Kubernetes Deployments

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
```

Kubernetes Deployments Updating

- Let’s update the nginx Pods to use the nginx:1.16.1 image instead of the nginx:1.14.2 image:

- kubectl --record deployment.apps/nginx-deployment set \
  image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.1
- kubectl set image deployment/nginx-deployment \
  nginx=nginx:1.16.1 –record
- kubectl edit deployment.v1.apps/nginx-deployment

Kubernetes Deployments

- Run kubectl get rs to see that the Deployment updated the Pods by creating a new ReplicaSet and scaling it up to 3 replicas, as well as scaling down the old ReplicaSet to 0 replicas.
- Running get pods should now show the Pods.

Kubernetes Deployments Scaling

- You can scale a Deployment by using the following command:
- kubectl scale deployment.v1.apps/nginx-deployment --replicas=10

- Assuming horizontal Pod autoscaling is enabled in your cluster, you can setup an autoscaler for your Deployment and choose the minimum and maximum number of Pods you want to run based on the CPU utilization of your existing Pods.
- kubectl autoscale deployment.v1.apps/nginx-deployment \

  --min=10 --max=15 --cpu-percent=80

Kubernetes Job and CronJob

- A Job creates one or more Pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created.
- A CronJob creates Jobs on a repeating schedule. CronJobs are useful for creating periodic and recurring tasks, like running backups or sending emails. CronJobs can also schedule individual tasks for a specific time, such as scheduling a Job for when your cluster is likely to be idle.

Kubernetes High-Level Controllers

- **Job**

- A Job is a supervisor for Pods that run a batch process. A Job creates Pods and ensures they do a task by tracking the number of successful Pod completions. Unlike a ReplicaSet, once the process inside the container finishes successfully, the container is not restarted. Use a Job when you want to run a process once.

Kubernetes Job

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Kubernetes High-Level Controllers

- **CronJob**

- If you want to run a Job at regular, specified times (e.g. hourly, daily, or monthly), create a CronJob. A CronJob is similar to a Job, but is scheduled to repeat at regular intervals or set times.

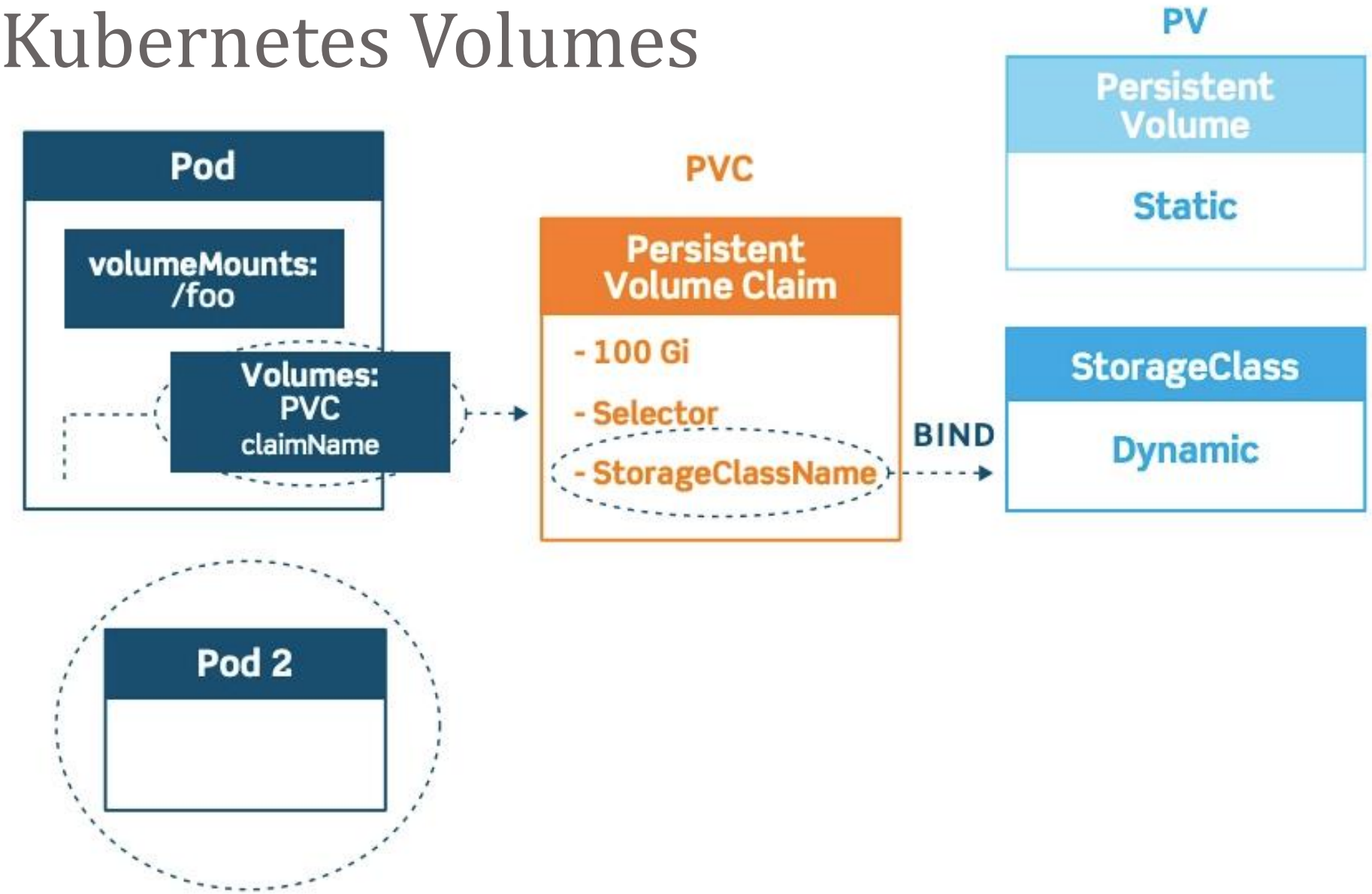
Kubernetes CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

Kubernetes Volumes

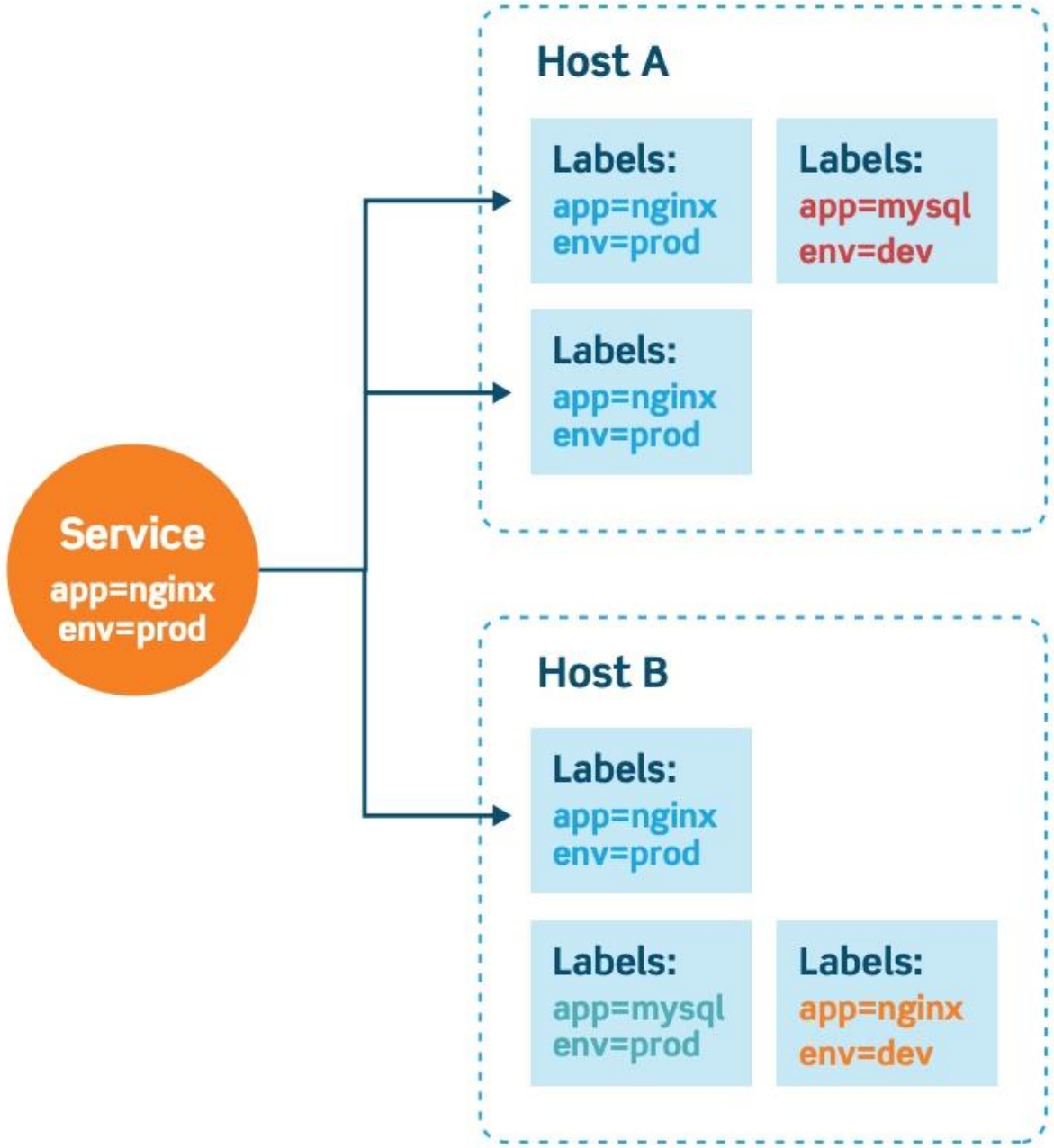
- A **Volume** is a directory that can hold data. A Volume is a component of a Pod, and is not independent of it. A Volume is created in the Pod specification. A Volume cannot be deleted on its own.
- A Volume is made accessible to all the containers in a Pod. Each container that you want to access the Volume must mount it individually.
- A K8s Volume outlives any individual containers, but when the enclosing Pod dies, the Volume dies, too. However, the files of some Volume types continue to exist in local or cloud storage, even after the Volume is gone.

Kubernetes Volumes



Kubernetes Service

- **Services** are the Kubernetes way of configuring a proxy to forward traffic to a set of pods.



Kubernetes Service Types

- **ClusterIP** exposes the service on an internal IP only. This makes the service reachable only from within the cluster. This is the default type.
- **NodePort** exposes the service on each node's IP at a specific port. This gives the developers the freedom to set up their own load balancers, for example, or configure environments not fully supported by Kubernetes.
- **LoadBalancer** exposes the service externally using a cloud provider's load balancer. This is often used when the cloud provider's load balancer is supported by Kubernetes, as it automates their configuration.
- **ExternalName** will just map a CNAME record in DNS. No proxying of any kind is established. This is commonly used to create a service within Kubernetes to represent an external datastore like a database that runs externally to Kubernetes.

Kubernetes

Environments

Kubernetes Environments

- **Learning environments**

Community	Ecosystem
Minikube	Docker Desktop
kind (Kubernetes IN Docker)	Minishift
	MicroK8s

Kubernetes Environments

- **Production environments**

Kubernetes Certified Service Providers

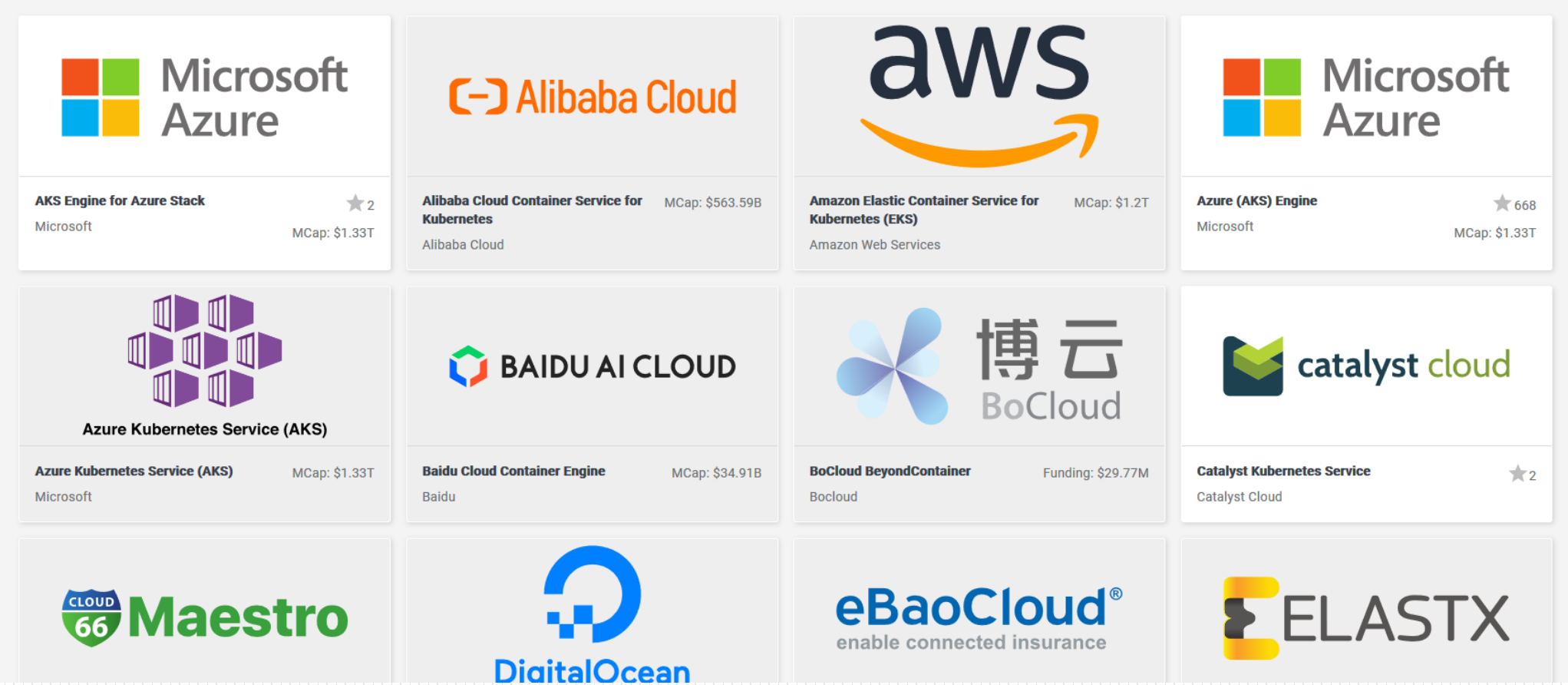
Vetted service providers with deep experience helping enterprises successfully adopt Kubernetes.

Certified Kubernetes Distributions, Hosted Platforms, and Installers

Software conformance ensures that every vendor's version of Kubernetes supports the required APIs.

Kubernetes Environments

- **Production environments**



Kubernetes

Minikube

Minikube: Getting Started

Minikube: Getting Started

- Start Minikube service

  minikube start --driver=<driver_name>

- Output:

  Starting local Kubernetes cluster...

  Running pre-create checks...

  Creating machine...

  Starting local Kubernetes cluster...

Minikube: Getting Started

- Deploy application

```
kubectl create deployment hello-minikube \
    --image=k8s.gcr.io/echoserver:1.10
```

- Output:

```
deployment.apps/hello-minikube created
```

Minikube: Getting Started

- Expose network port

```
kubectl expose deployment hello-minikube \
    --type=NodePort --port=8080
```

- Output:

```
service/hello-minikube exposed
```

Minikube: Getting Started

- View pod information

  kubectl get pod

- Output:

```
NAME                              READY   STATUS              RESTARTS   AGE
hello-minikube-3383150820-vctvh   0/1     ContainerCreating   0          3s
```

Minikube: Getting Started

- Delete application

  kubectl delete services hello-minikube

- Output:

  service "hello-minikube" deleted

Minikube: Getting Started

- Delete deployment

  kubectl delete deployment hello-minikube

- Output:

  deployment.extensions "hello-minikube" deleted

Kubernetes

Kubeadm

Kubeadm: Getting Started

Kubeadm: Getting Started

- Create new deployment

  kubeadm init

- Output:

  Your Kubernetes control-plane has initialized successfully!

Kubeadm: Getting Started

- Add additional nodes

```
kubeadm join --token <token> \
    <control-plane-host>:<control-plane-port> \
    --discovery-token-ca-cert-hash sha256:<hash>
```

Kubeadm: Getting Started

- Show tokens

  kubeadm token list

- Output:

TOKEN	TTL	EXPIRES	USAGES	DESCRIPTION	EXTRA GROUPS
8ewj1p.9r9hcjoqgajrj4gi	23h	2018-06-12T02:51:28Z	authentication, signing	The default bootstrap token generated by 'kubeadm init'.	system: bootstrappers: kubeadm: default-node-token

Kubeadm: Getting Started

- Remove nodes (partially)

```
kubectl config delete-cluster
```

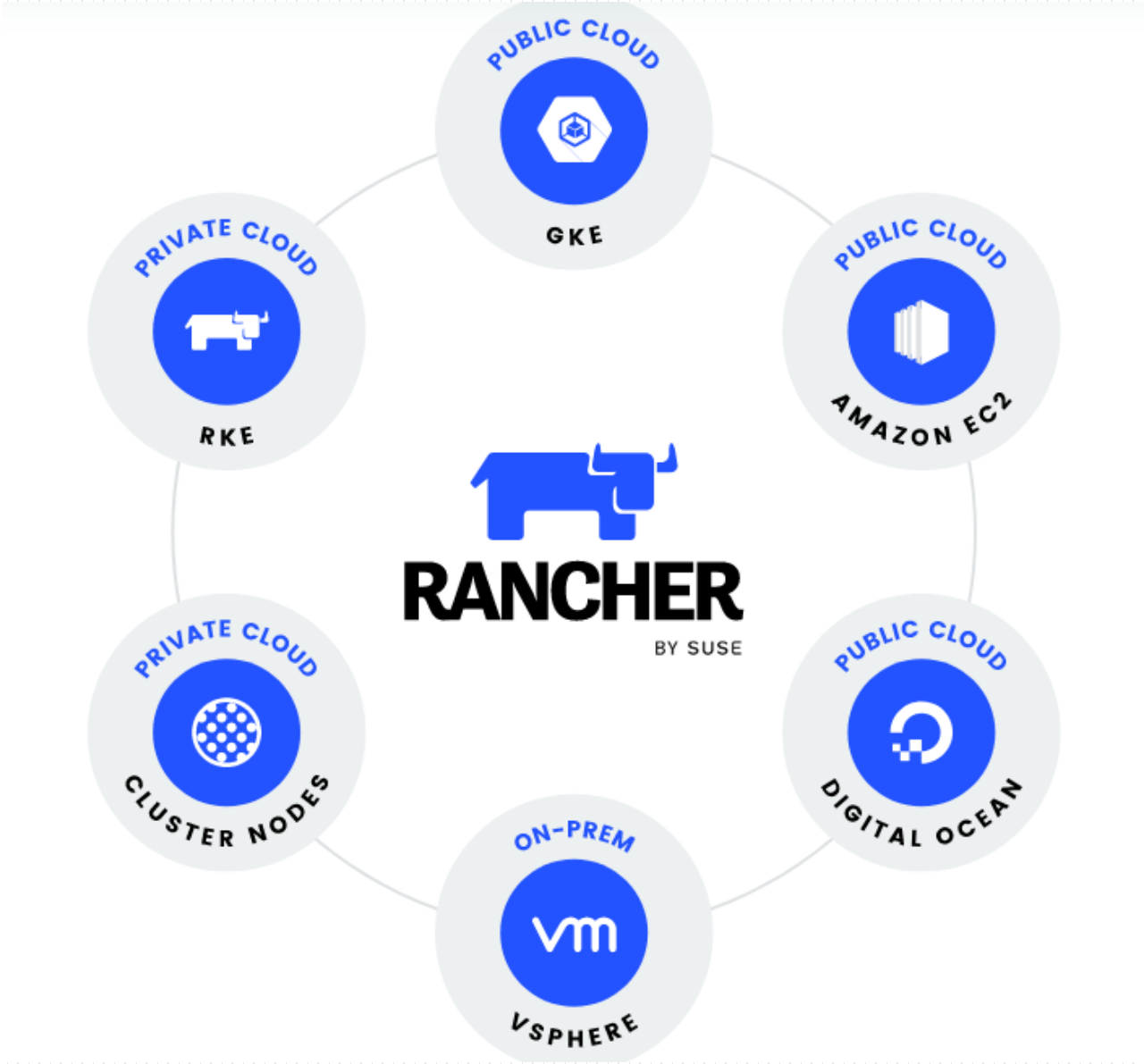
- Remove nodes (more cleanly)

```
kubectl drain <node name> --delete-local-data \
        --force --ignore-daemonsets
kubectl delete node <node name>
```

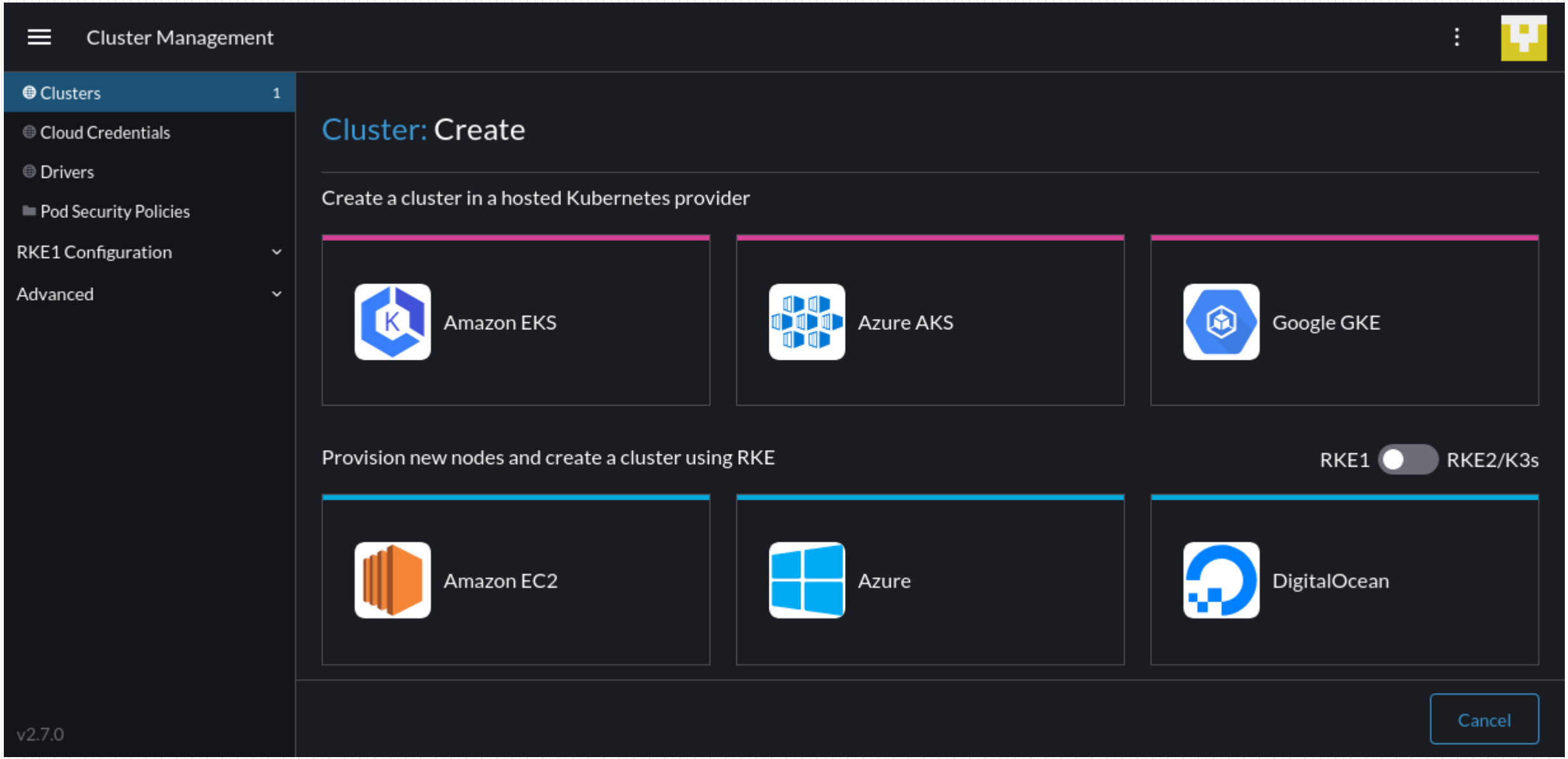
Kubernetes

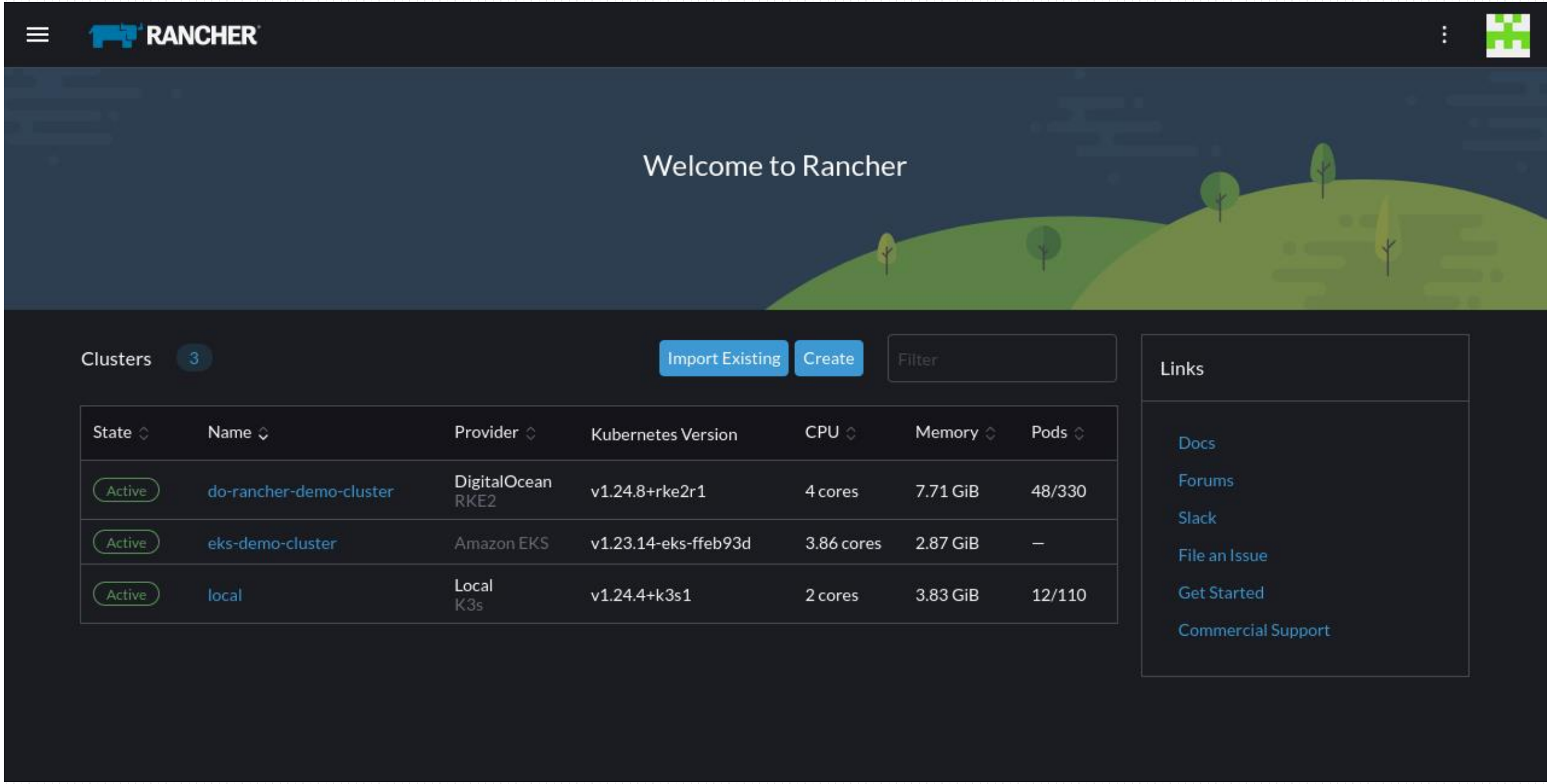
Rancher

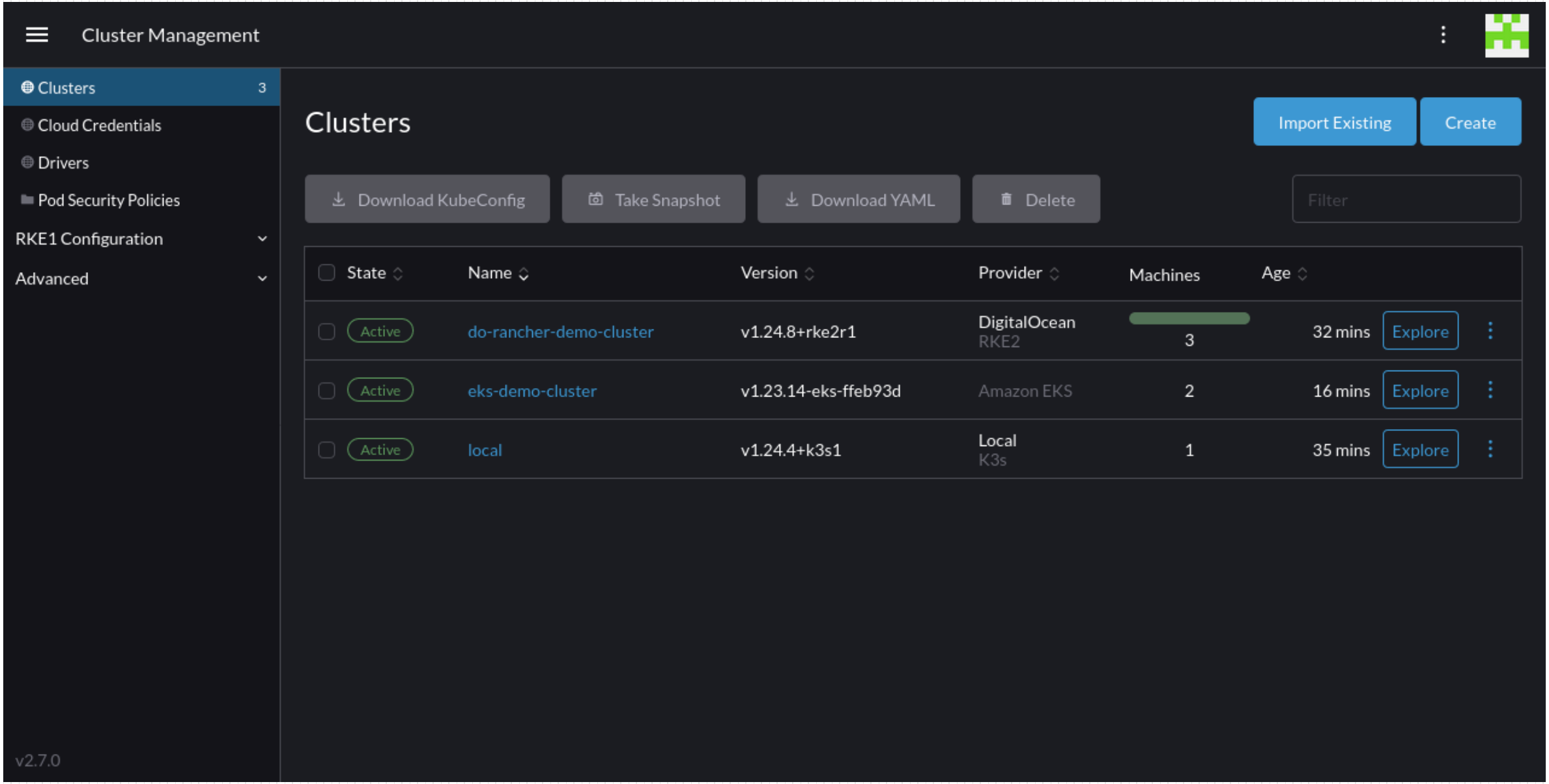
Rancher:
Getting Started

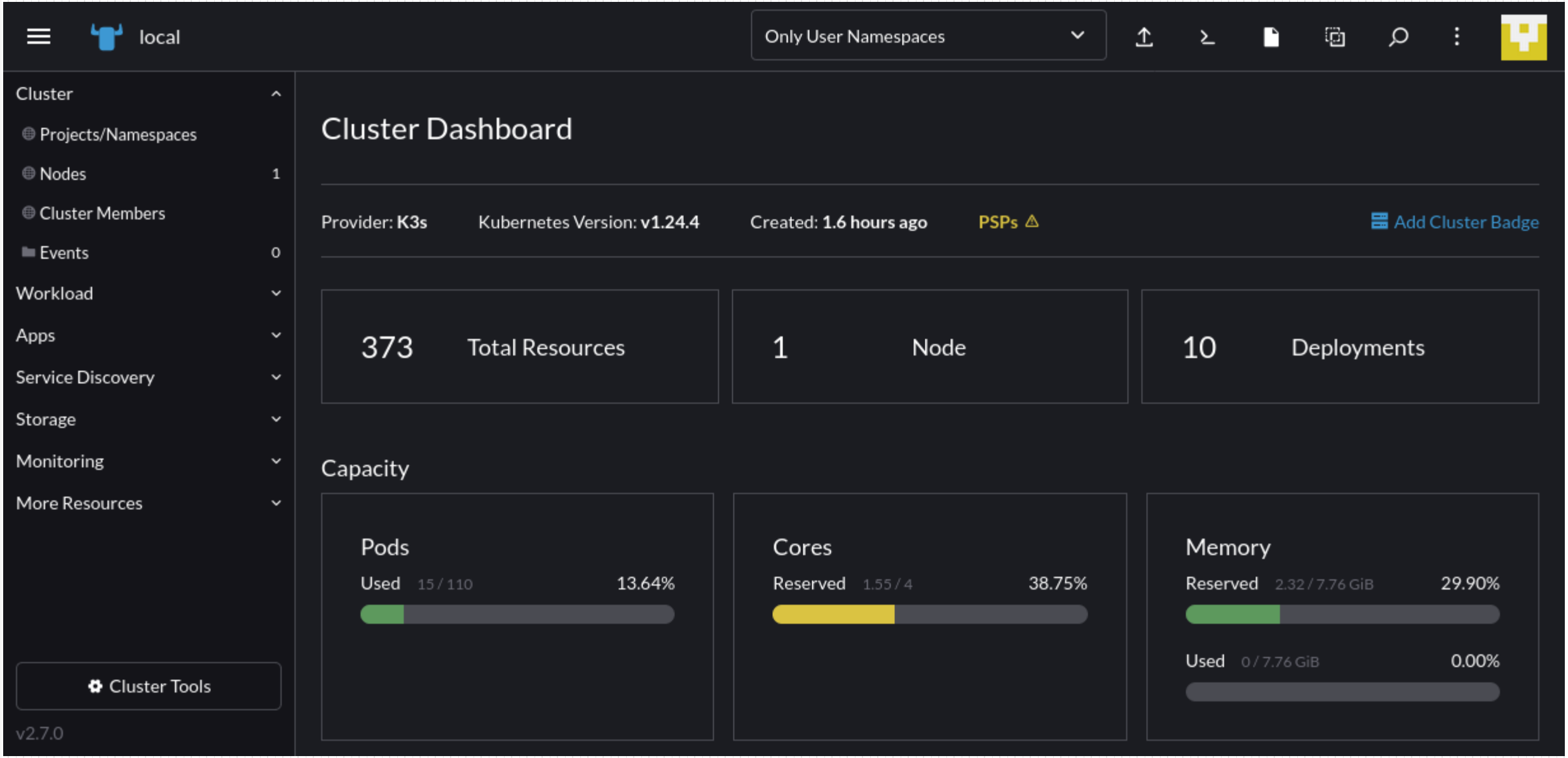


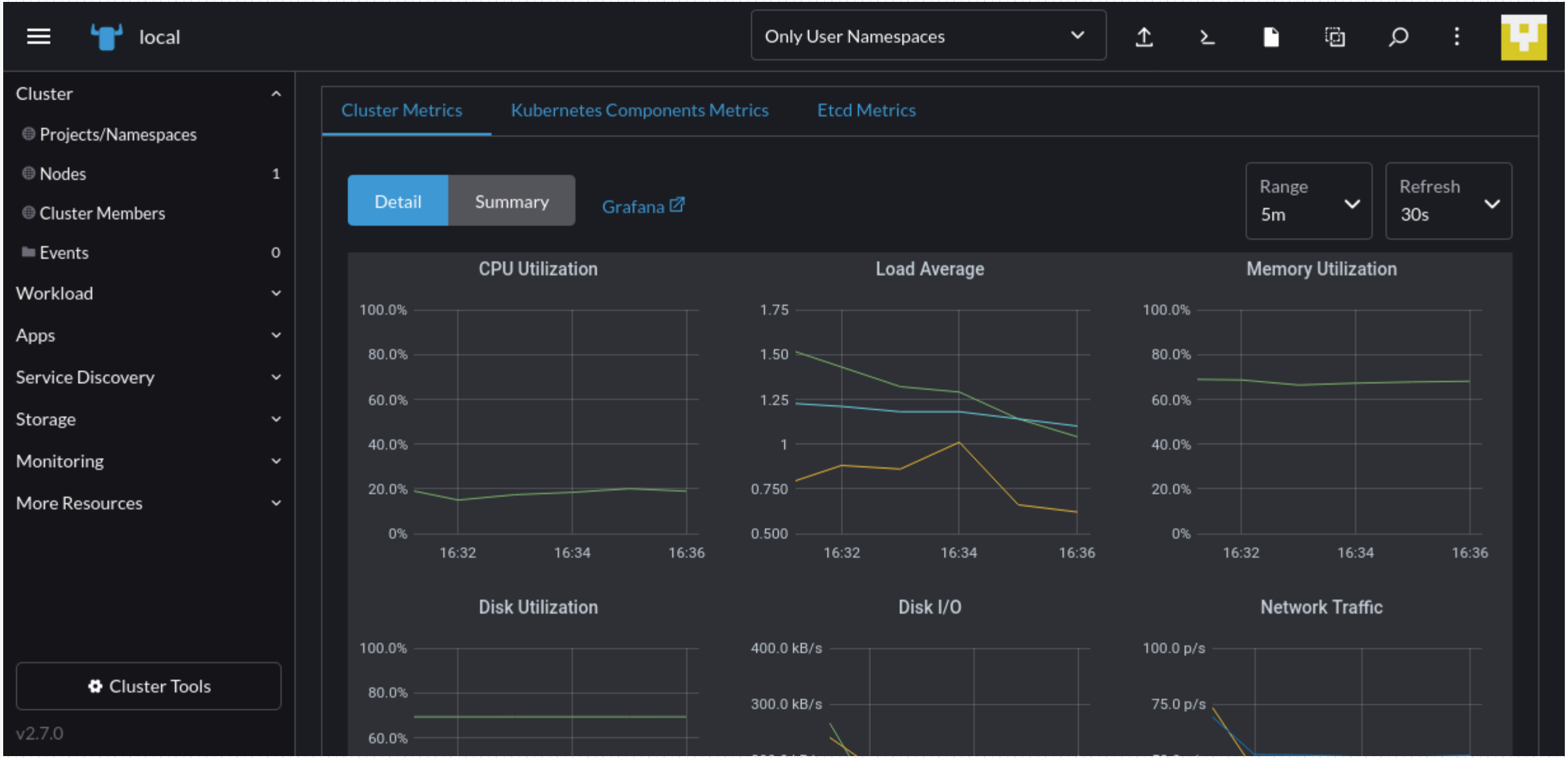
Rancher:
Getting Started

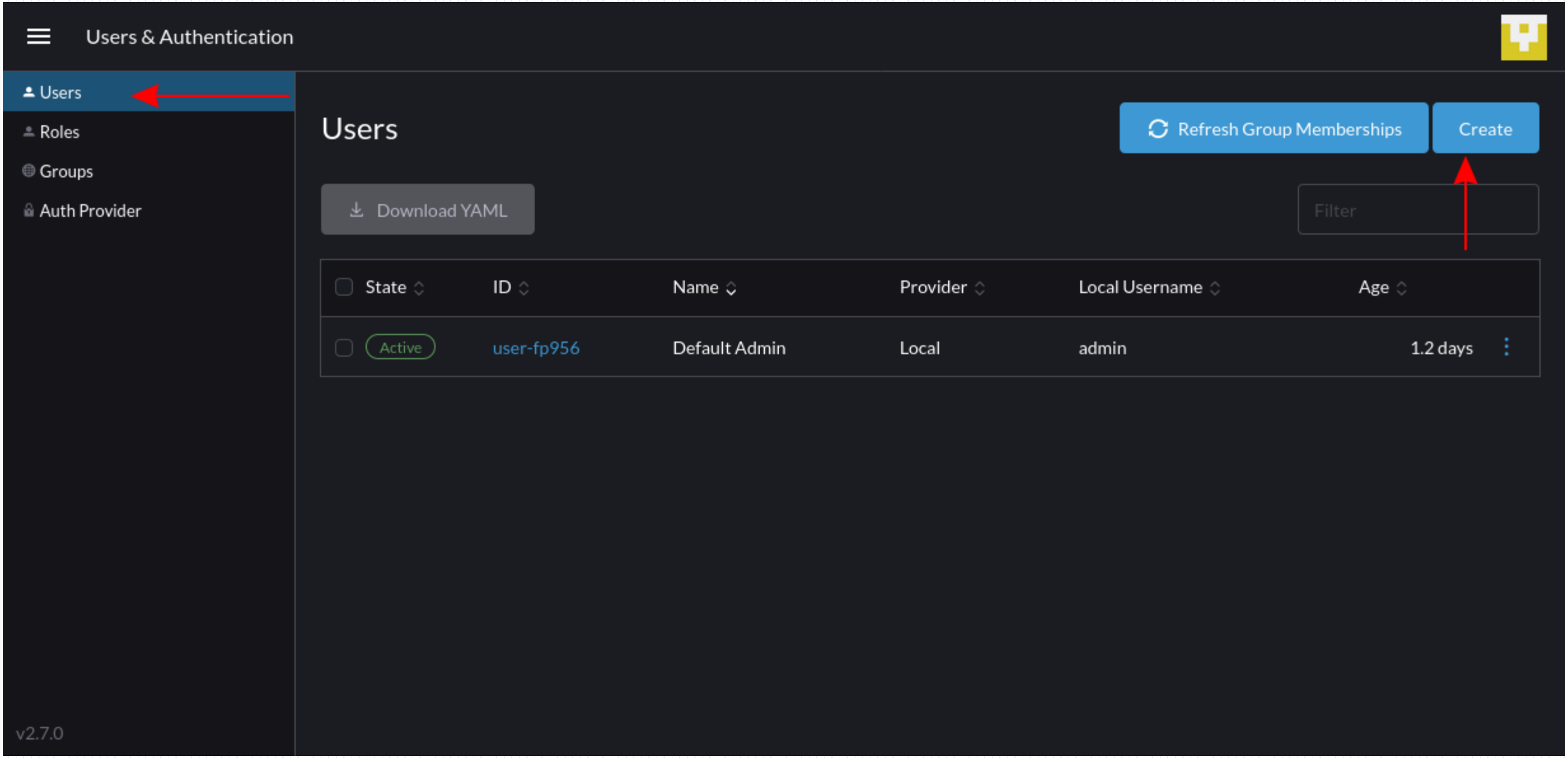

Cluster Management
Clusters 1
Cloud Credentials
Drivers
Pod Security Policies
RKE1 Configuration
Advanced
v2.7.0
Cluster: Create
Create a cluster in a hosted Kubernetes provider
Amazon EKS
Azure AKS
Google GKE
Provision new nodes and create a cluster using RKE
RKE1
RKE2/K3s
Amazon EC2
Azure
DigitalOcean
Cancel

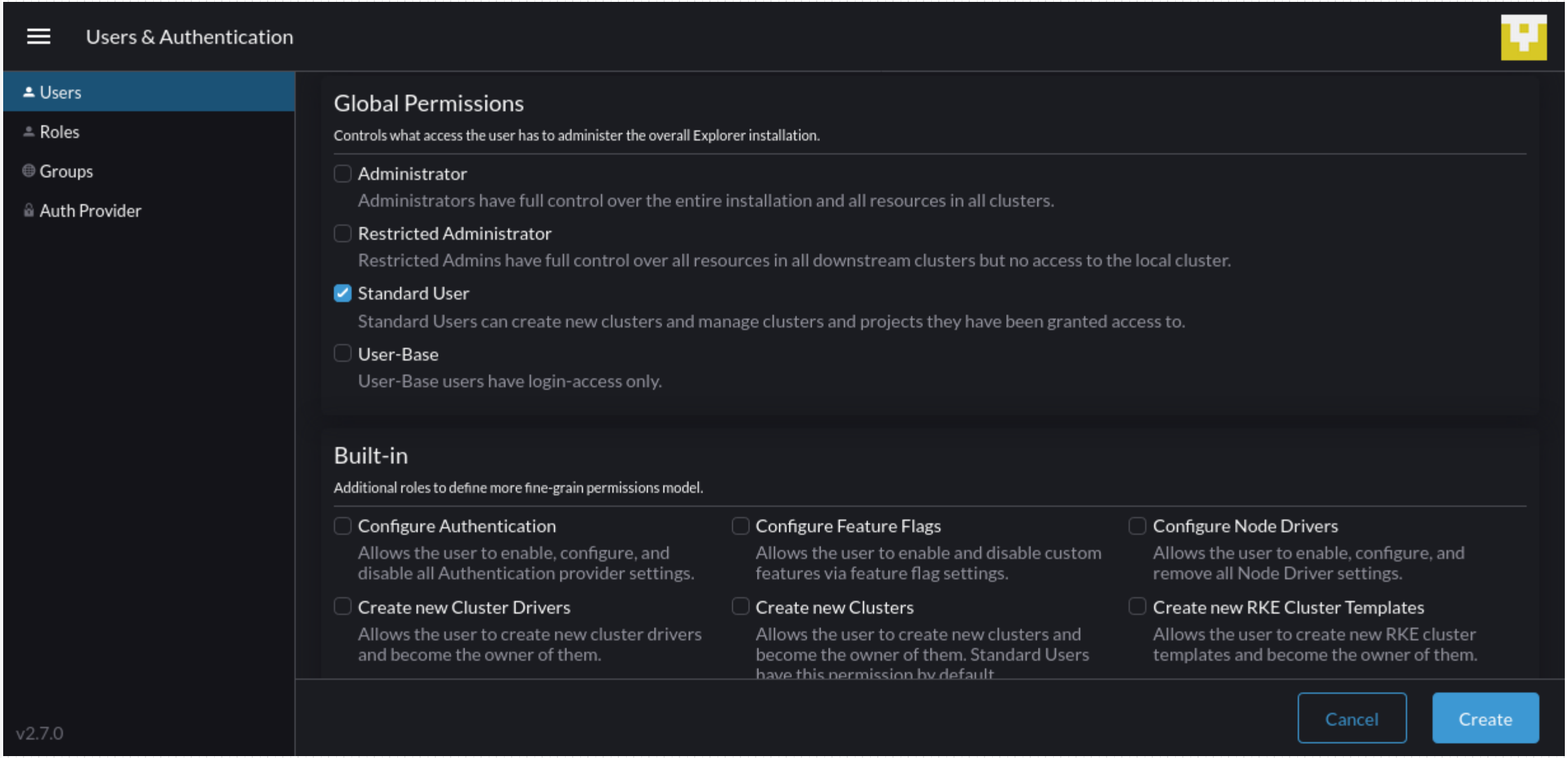

RANCHER
Welcome to Rancher
Clusters 3
Import Existing
Create
Filter
State
Name
Provider
Kubernetes Version
CPU
Memory
Pods
Active
do-rancher-demo-cluster
DigitalOcean
RKE2
v1.24.8+rke2r1
4 cores
7.71 GiB
48/330
Active
eks-demo-cluster
Amazon EKS
v1.23.14-eks-ffeb93d
3.86 cores
2.87 GiB
—
Active
local
Local
K3s
v1.24.4+k3s1
2 cores
3.83 GiB
12/110
Links
Docs
Forums
Slack
File an Issue
Get Started
Commercial Support

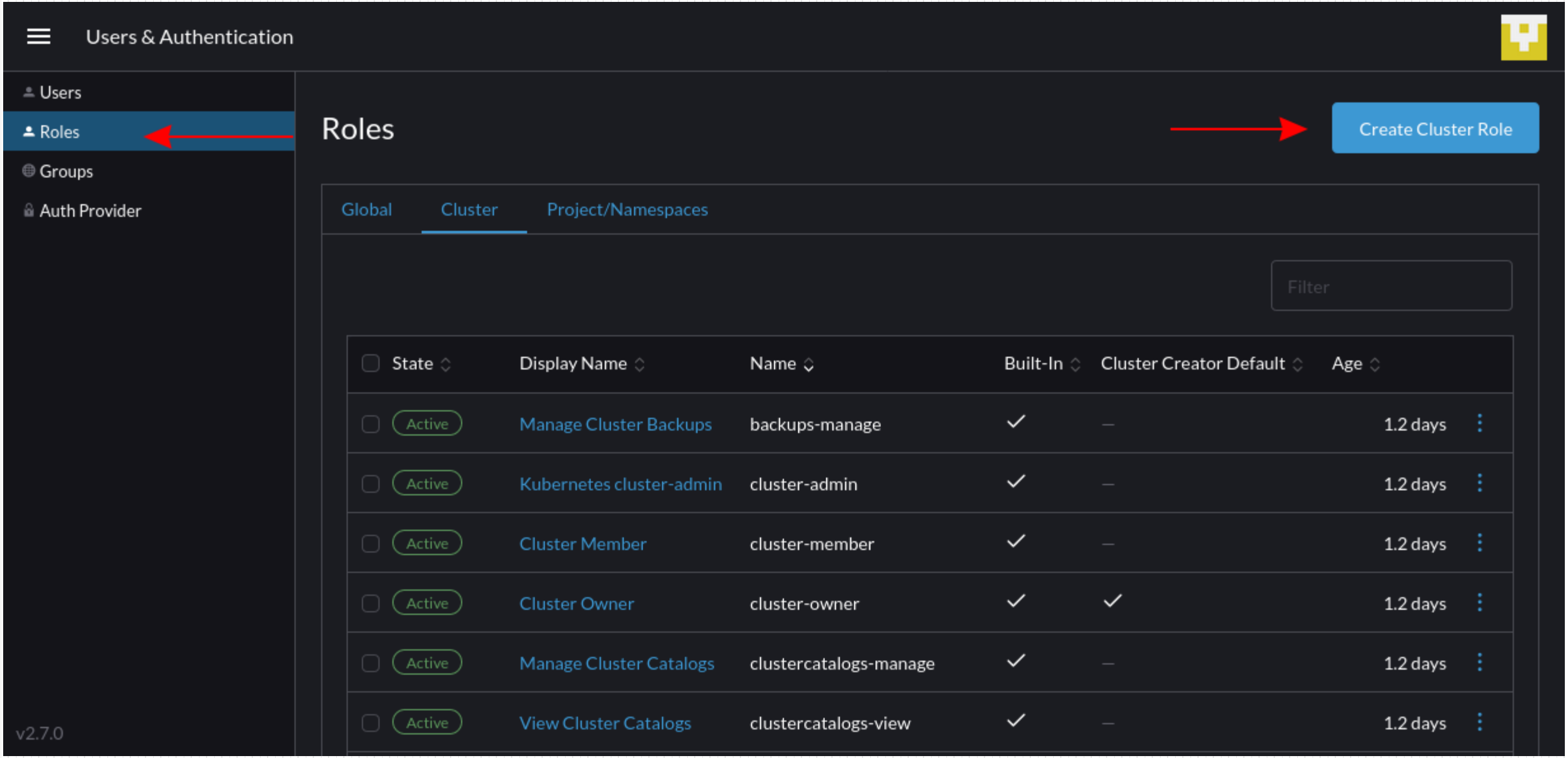

Cluster Management
Clusters 3
Cloud Credentials
Drivers
Pod Security Policies
RKE1 Configuration
Advanced
v2.7.0
Clusters
Import Existing
Create
Download KubeConfig
Take Snapshot
Download YAML
Delete
Filter
State
Name
Version
Provider
Machines
Age
Active
do-rancher-demo-cluster
v1.24.8+rke2r1
DigitalOcean
RKE2
3
32 mins
Explore
Active
eks-demo-cluster
v1.23.14-eks-ffeb93d
Amazon EKS
2
16 mins
Explore
Active
local
v1.24.4+k3s1
Local
K3s
1
35 mins
Explore

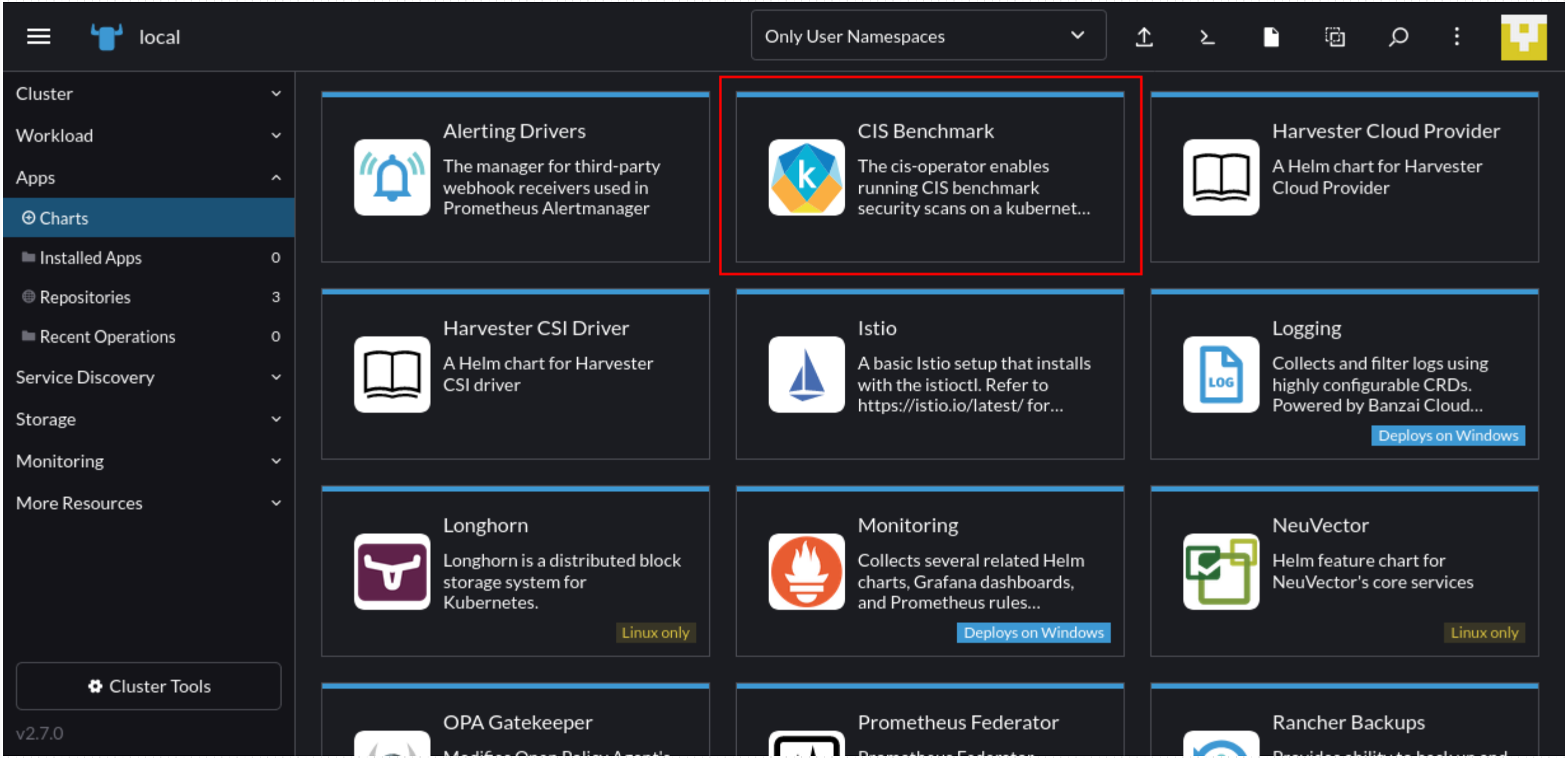

local
Only User Namespaces
Cluster
Projects/Namespaces
Nodes 1
Cluster Members
Events 0
Workload
Apps
Service Discovery
Storage
Monitoring
More Resources
Cluster Tools
v2.7.0
Cluster Dashboard
Provider: K3s
Kubernetes Version: v1.24.4
Created: 1.6 hours ago
PSPs
Add Cluster Badge
373 Total Resources
1 Node
10 Deployments
Capacity
Pods
Used 15 / 110 13.64%
Cores
Reserved 1.55 / 4 38.75%
Memory
Reserved 2.32 / 7.76 GiB 29.90%
Used 0 / 7.76 GiB 0.00%

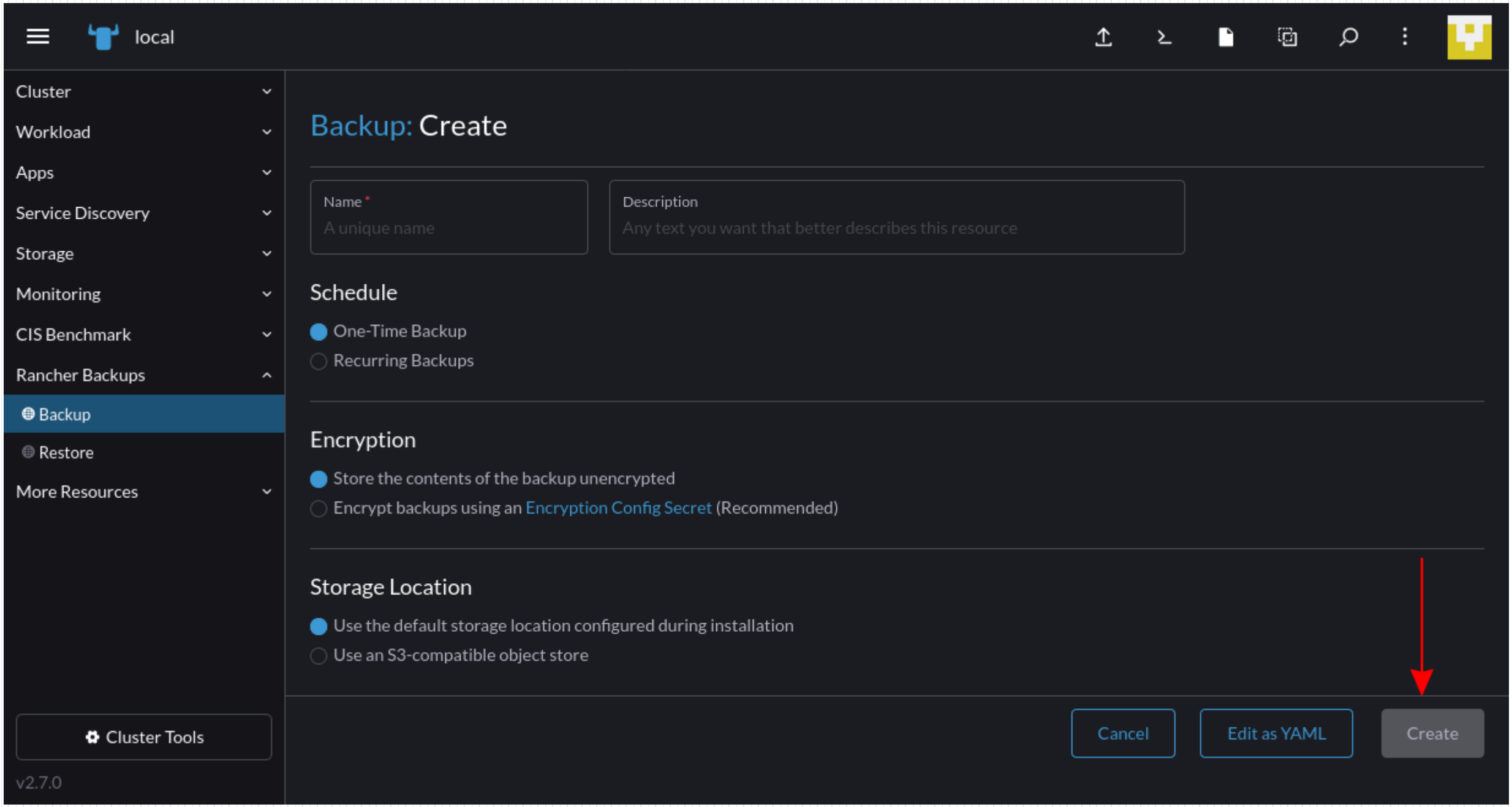

local
Only User Namespaces
Cluster
Projects/Namespaces
Nodes 1
Cluster Members
Events 0
Workload
Apps
Service Discovery
Storage
Monitoring
More Resources
Cluster Tools
v2.7.0
Cluster Metrics
Kubernetes Components Metrics
Etcd Metrics
Detail
Summary
Grafana
Range 5m
Refresh 30s
CPU Utilization
100.0%
80.0%
60.0%
40.0%
20.0%
0%
16:32
16:34
16:36
Load Average
1.75
1.50
1.25
1
0.750
0.500
Memory Utilization
Disk Utilization
Disk I/O
400.0 kB/s
300.0 kB/s
Network Traffic
100.0 p/s
75.0 p/s

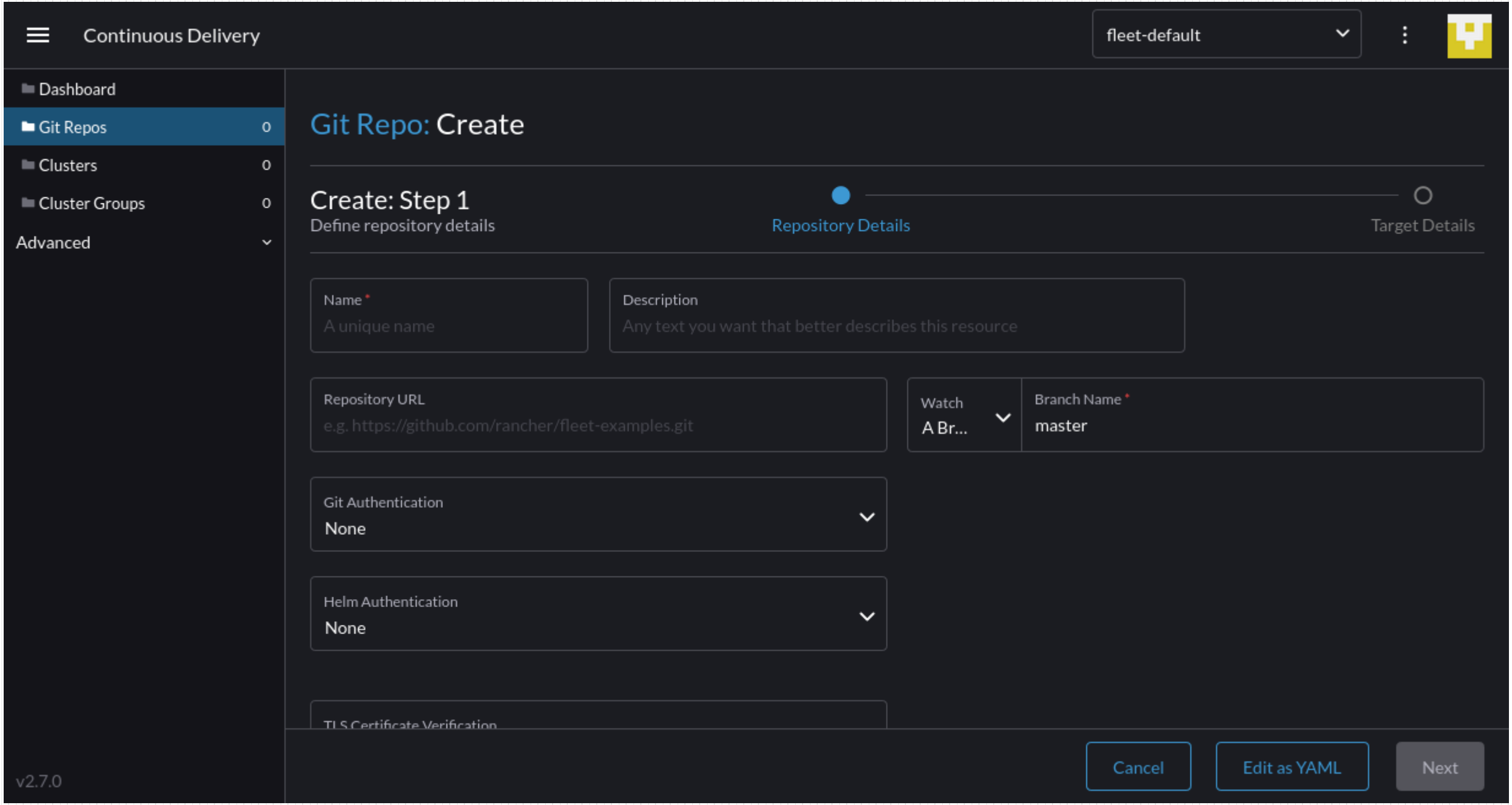

Users & Authentication
Users
Roles
Groups
Auth Provider
v2.7.0
Users
Refresh Group Memberships
Create
Download YAML
Filter
State
ID
Name
Provider
Local Username
Age
Active
user-fp956
Default Admin
Local
admin
1.2 days


Users & Authentication
Users
Roles
Groups
Auth Provider
v2.7.0
Global Permissions
Controls what access the user has to administer the overall Explorer installation.
Administrator
Administrators have full control over the entire installation and all resources in all clusters.
Restricted Administrator
Restricted Admins have full control over all resources in all downstream clusters but no access to the local cluster.
Standard User
Standard Users can create new clusters and manage clusters and projects they have been granted access to.
User-Base
User-Base users have login-access only.
Built-in
Additional roles to define more fine-grain permissions model.
Configure Authentication
Allows the user to enable, configure, and disable all Authentication provider settings.
Configure Feature Flags
Allows the user to enable and disable custom features via feature flag settings.
Configure Node Drivers
Allows the user to enable, configure, and remove all Node Driver settings.
Create new Cluster Drivers
Allows the user to create new cluster drivers and become the owner of them.
Create new Clusters
Allows the user to create new clusters and become the owner of them. Standard Users
Create new RKE Cluster Templates
Allows the user to create new RKE cluster templates and become the owner of them.
Cancel
Create


Users & Authentication
Users
Roles
Groups
Auth Provider
v2.7.0
Roles
Create Cluster Role
Global
Cluster
Project/Namespaces
Filter
State
Display Name
Name
Built-In
Cluster Creator Default
Age
Active
Manage Cluster Backups
backups-manage
1.2 days
Active
Kubernetes cluster-admin
cluster-admin
1.2 days
Active
Cluster Member
cluster-member
1.2 days
Active
Cluster Owner
cluster-owner
1.2 days
Active
Manage Cluster Catalogs
clustercatalogs-manage
1.2 days
Active
View Cluster Catalogs
clustercatalogs-view
1.2 days


local
Only User Namespaces
Cluster
Workload
Apps
Charts
Installed Apps 0
Repositories 3
Recent Operations 0
Service Discovery
Storage
Monitoring
More Resources
Cluster Tools
v2.7.0
Alerting Drivers
The manager for third-party webhook receivers used in Prometheus Alertmanager
CIS Benchmark
The cis-operator enables running CIS benchmark security scans on a kubernet...
Harvester Cloud Provider
A Helm chart for Harvester Cloud Provider
Harvester CSI Driver
A Helm chart for Harvester CSI driver
Istio
A basic Istio setup that installs with the istioctl. Refer to https://istio.io/latest/ for...
Logging
Collects and filter logs using highly configurable CRDs. Powered by Banzai Cloud...
Deploys on Windows
Longhorn
Longhorn is a distributed block storage system for Kubernetes.
Linux only
Monitoring
Collects several related Helm charts, Grafana dashboards, and Prometheus rules...
Deploys on Windows
NeuVector
Helm feature chart for NeuVector's core services
Linux only
OPA Gatekeeper
Prometheus Federator
Rancher Backups


local
Cluster
Workload
Apps
Service Discovery
Storage
Monitoring
CIS Benchmark
Rancher Backups
Backup
Restore
More Resources
Cluster Tools
v2.7.0
Backup: Create
Name *
A unique name
Description
Any text you want that better describes this resource
Schedule
One-Time Backup
Recurring Backups
Encryption
Store the contents of the backup unencrypted
Encrypt backups using an Encryption Config Secret (Recommended)
Storage Location
Use the default storage location configured during installation
Use an S3-compatible object store
Cancel
Edit as YAML
Create


Continuous Delivery
fleet-default
Dashboard
Git Repos 0
Clusters 0
Cluster Groups 0
Advanced
v2.7.0
Git Repo: Create
Create: Step 1
Define repository details
Repository Details
Target Details
Name *
A unique name
Description
Any text you want that better describes this resource
Repository URL
e.g. https://github.com/rancher/fleet-examples.git
Watch
A Br...
Branch Name *
master
Git Authentication
None
Helm Authentication
None
TLS Certificate Verification
Cancel
Edit as YAML
Next

Questions?

andry.cheredarchuk@gmail.com