1. General info

Welcome to the workshop! The goal of this guide is to walk you through the deployment of Rancher, two Kubernetes clusters, Longhorn and a couple applications. This is not the definitive guide to Kubernetes, but will at a minimum serve as a way to build a cluster which can be used for application development, testing, and production (with higher VM specifications). One Rancher cluster can support multiple worker clusters. We will be utilizing Ubuntu, but clusters can be of a variety of Linux distributions and in specific cases Windows. Windows is out of scope for this workshop and not recommended for this process in general. A better practice is to build for a native Linux deployment, in our case .Net Core, and avoid the Windows overhead and complexity for containers. Use the right tool for the job.

We will be connecting to the servers (soon to become nodes!) using SSH. I recommend using Putty. OpenSSH can be launched via PowerShell as well in Windows 10 and Windows 11 as well, but is less friendly.

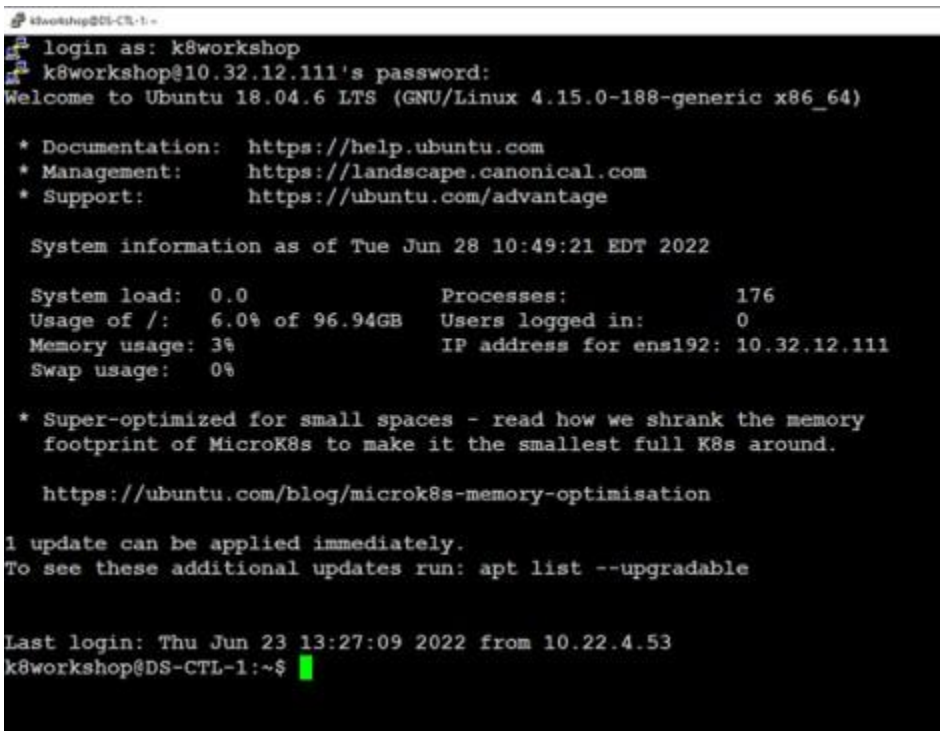
The credentials for the workshop are provided below. They are the same for all participants.The password is case sensitive.

```
User: k8workshop
Password: WorkingK8!!
```

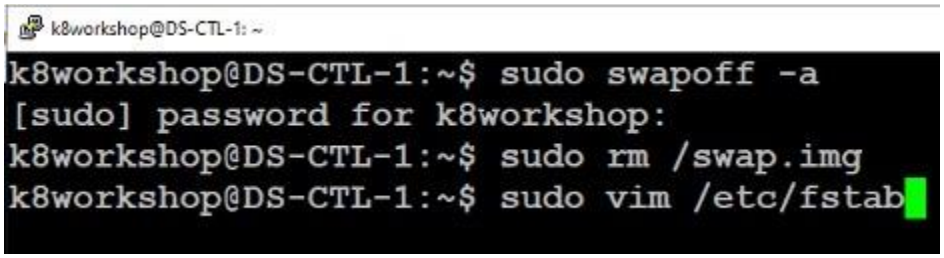
2. Prepare the control cluster

The control cluster is what hosts the Rancher application and monitoring for Rancher. That is the sole purpose. Any actual workloads will operate on the Worker cluster(s). For a Rancher deployment in our environment, we will use the RKE2 install, Rancher Kubernetes Engine 2, to host the Rancher instance on a 3 node cluster.

1. SSH into your 1st Node of the Control Plan cluster. This should be {Initials}-CTL-1.
2. If prompted for Yes/No of a SSL thumbprint, please accept



3. Disable swap and then reboot. This can be done with either Vim or Nano




```
sudo swapoff -a
sudo rm /swap.img
```

4. Edit /etc/fstab

a. Using Vim

```
sudo vim /etc/fstab
```

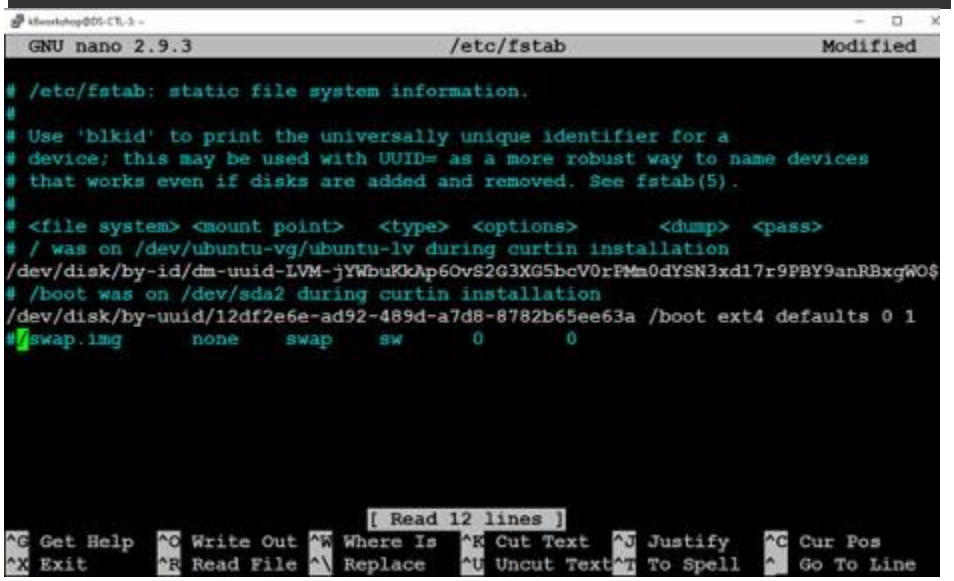
Insert a # before the /swap. Press i to enter –InsertMode–

Press “Escape colon w q” then enter

b. Or using Nano just type # in front of the /swap

Then control+x, y, Enter.

```
sudo nano /etc/fstab
```



5. Run this command to verify that the swap is off. There should be no output; a direct return to prompt.

```
k8workshop@DS-CTL-1:~$ sudo swapon --show
```

```
sudo swapon --show
```

6. Reboot the VM using:

```
sudo reboot
```

7. Repeat steps 1 through 6 with your CTL-2 and CTL-3 servers, as well as WRK-1 through WRK-3.

3. Deploy the control cluster

Now that we have disabled swap on the 3 control plane and 3 worker nodes, we can deploy Rke2 Kubernetes to the control cluster.

1. Become root:

```
sudo -i
```

2. Download the Rancher Kubernetes Engine 2 (RKE2) installation script and deploy.

```
curl -sfL https://get.rke2.io | sh -
```

3. Create a path for the configuration file.

```
mkdir -p /etc/rancher/rke2/
```

4. Configure the config.yaml for rke2

**** Please note the first rancher node needs to have the server line commented out. ***

```
vim /etc/rancher/rke2/config.yaml
```

Contents of the yaml file:

```
token: k8workshop22
tls-san:
  - kubeapi-{YourInitialsHere}.some.domain
#server: https://kubeapi-{YourInitialsHere}.some.domain:9345
```

On nodes CTL-2 and CTL-3:

```
token: k8workshop22
tls-san:
  - kubeapi-{YourInitialsHere}.some.domain
server: https://kubeapi-{YourInitialsHere}.some.domain:9345
```

Comments:

The token represents your cluster password for them to know it's safe to talk to the other. It's used for bootstrapping as well. The tls-san is the DNS name for your Kubernetes API endpoint. In a production scenario it will be load balanced across the control cluster. This is part of the lifeblood of your environment and must be highly available.

5. Start the cluster service on CTL-1.

```
systemctl enable rke2-server.service
systemctl start rke2-server.service
```

Comments:

It can take a minute or two for the service to start up. If everything goes as planned, you will see a return to the prompt and no error messages. If you encounter an error, it will say something about "`journalctl -xe`". You would then need to use that command to start troubleshooting the issue.

6. Execute the same systemctl enable and start commands on CTL-2 and CTL-3
7. Execute a kubectl command to verify that RKE2 has started on all 3 nodes:

```
/var/lib/rancher/rke2/bin/kubectl --kubeconfig
/etc/rancher/rke2/rke2.yaml get nodes
```

8. The get nodes output should look like the following once all nodes areresponding or bootstrapping.

```
root@DS-CTL-1:~# /var/lib/rancher/rke2/bin/kubectl --kubeconfig /etc/rancher/rke2/rke2.yaml get nodes
NAME       STATUS     ROLES                       AGE   VERSION
ds-ctl-1   Ready      control-plane,etcd,master   52s   v1.23.7+rke2r2
ds-ctl-2   Ready      control-plane,etcd,master   36s   v1.23.7+rke2r2
ds-ctl-3   NotReady   control-plane,etcd,master   10s   v1.23.7+rke2r2
root@DS-CTL-1:~#
```

9. Perform an export to add RKE2's bin folder to PATH

```
export PATH=$PATH:/var/lib/rancher/rke2/bin/
```

10. On CTL-2 and CTL-3 perform the following command to create a necessaryfolder:

```
mkdir /root/.kube
```

11. On all 3 nodes perform this command to persist the RKE2 config into a moreconvenient folder for Kubectl access.

```
cp /etc/rancher/rke2/rke2.yaml /root/.kube/config
```

4. Configure the pre-requisites and deploy Rancher

1. Deploy Helm 3, which is the core component for deploying Kubernetes applications (Helm Charts).

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

2. Add the jetstack certificate manager repository

```
helm repo add jetstack https://charts.jetstack.io
```

3. Add the Rancher repository (stable release)

```
helm repo add rancher-stable https://releases.rancher.com/server-charts/stable
```

4. Fetch the two container images (Jetstack and Rancher).

```
helm fetch jetstack/cert-manager --version v1.8.2
```

```
helm fetch rancher-stable/rancher --version 2.6.5
helm repo update
```

5. Install the Cert Manager image (Helm Chart)

```
helm install \ cert-manager
    jetstack/cert-manager \ --namespace
    cert-manager \
    --create-namespace \
    --version v1.8.2 \
    --set installCRDs=true
```

6. Create the namespace for Rancher

```
kubectl create ns cattle-system
```

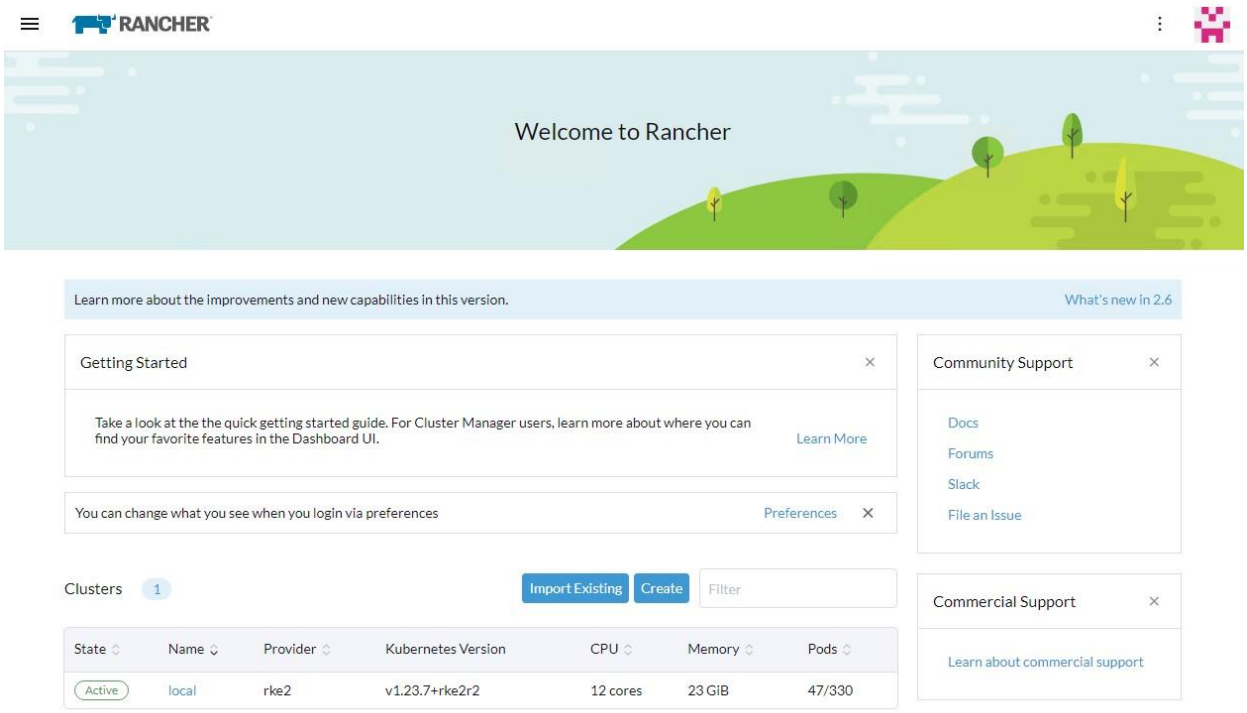
7. Install the actual Rancher application via the image (Helm Chart)

```
helm install rancher rancher-stable/rancher \
    --namespace cattle-system \
    --set hostname=rancher-ds.some.domain \
    --version 2.6.5 \
    --set bootstrapPassword=k8workshopboot
```

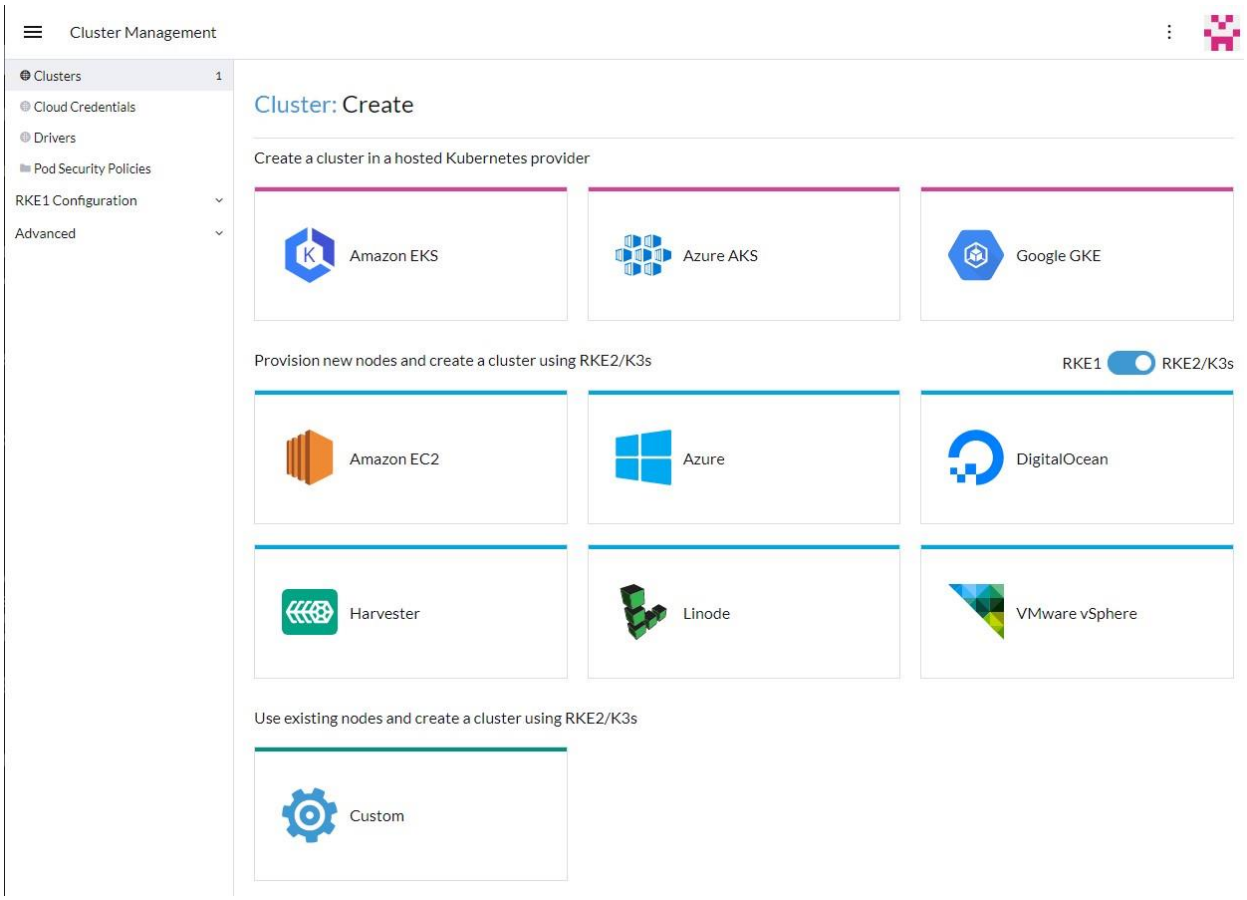
8. The bootstrap password is configured here as k8workshopboot. You would wantsomething a little more secure in a production environment. The purpose is for initial admin password during deployment. This is the password you will use for initial login to Rancher.

5. Accessing Rancher

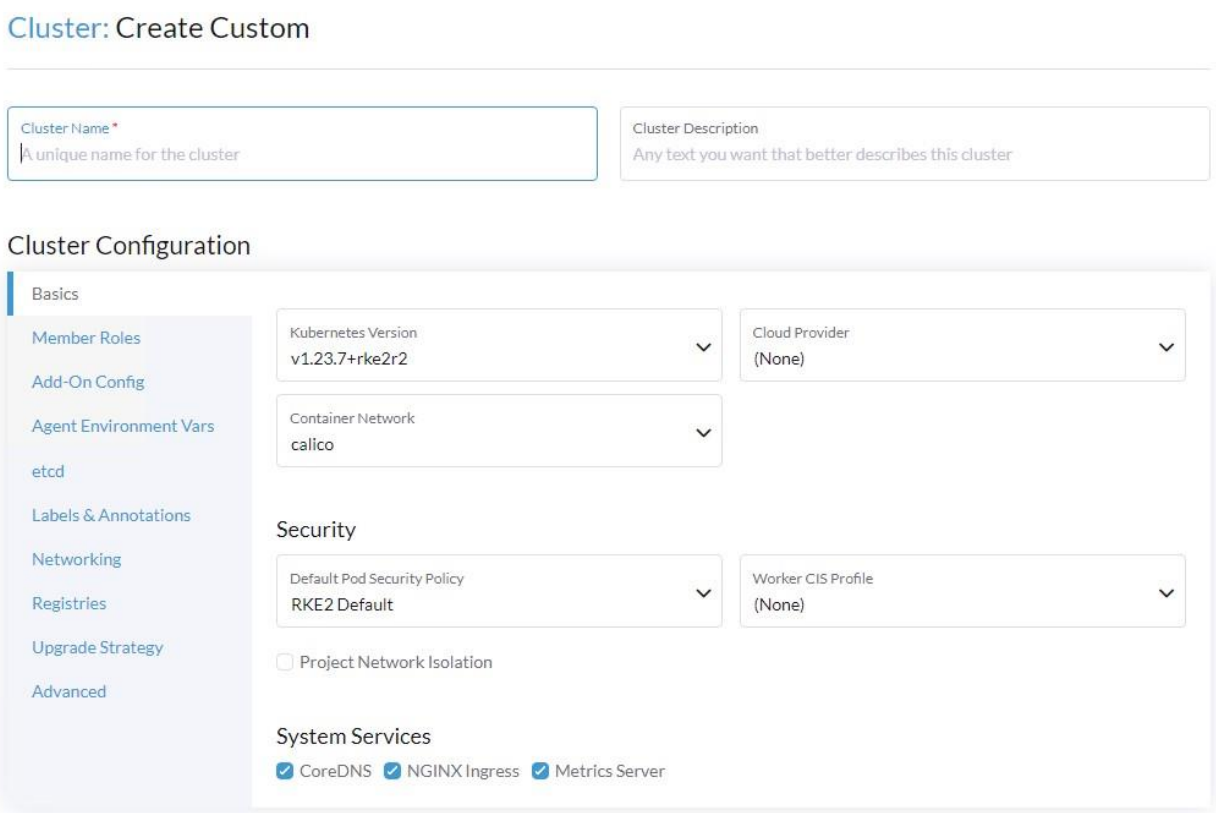
Once you complete the prior section, your Rancher install should be online. Now we will proceed to get into the Rancher user interface and use it to deploy our worker cluster. A production Rancher environment has one cluster dedicated to Rancher.



We're going to start with the Create button directly above the existing "local" cluster, which is also our Rancher host cluster.

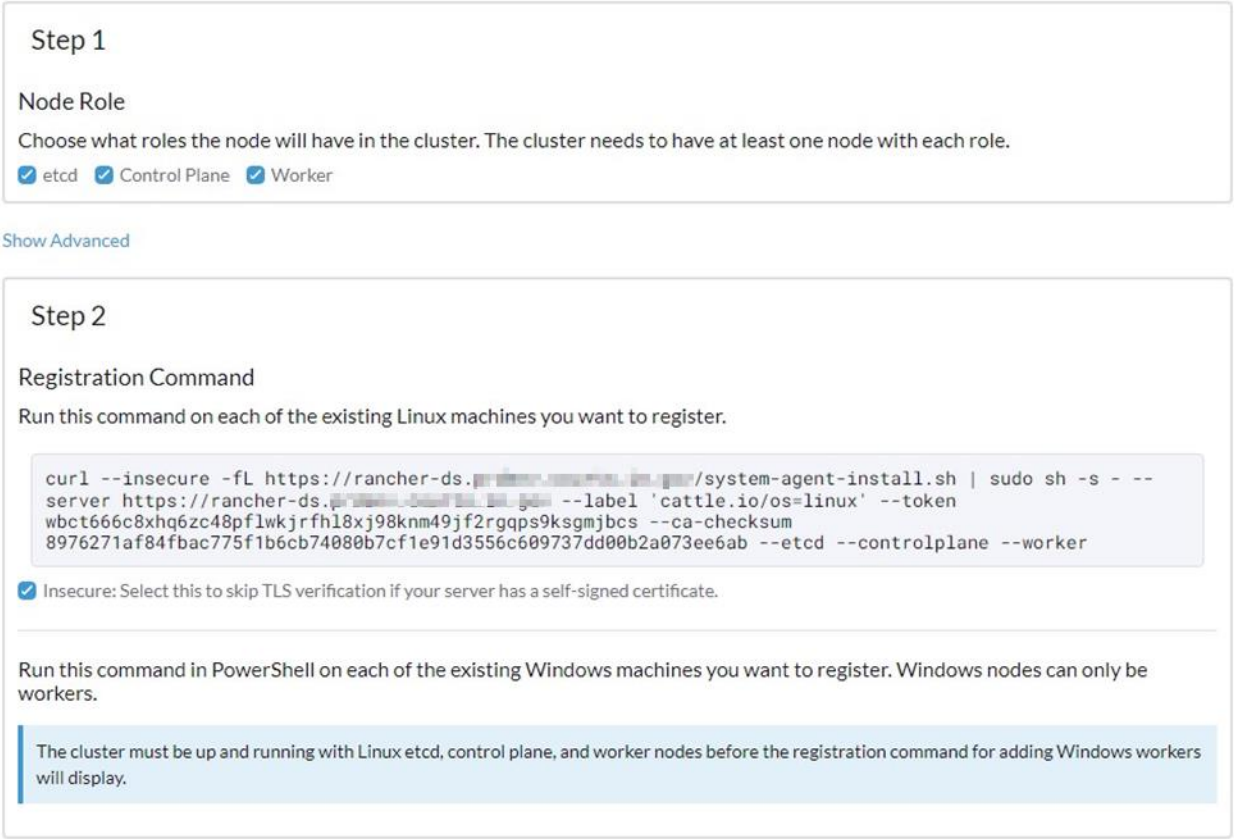


From here we will turn ON the RKE1 vs RKE2/K3s toggle such that it turns blue. That will give us the option for Custom at the bottom and the ability to deploy a worker cluster (and expand it) with a simple command.



We will actually use the default configuration for most steps. There are a lot of things that can be customized, but are not specifically relevant here. We must enter a cluster name (all lowercase) and optionally a description. Here is a valid example:

We will then want to configure the Registration tab of the worker cluster.



In Step 1., select all 3 roles. It does no harm for these to be present on every node. You also must have at least 1 node with each role for the cluster to start properly. I**f you neglect to pick all 3 for the initial script, you will need to revert to a snapshot and redeploy WRK-1 through WRK-3.**

In Step 2., be sure to CHECK the Insecure: box. That will enable the cluster to use a self-signed certificate.
You will then copy and paste the command in the code box. It is specific to each cluster, so I cannot provide it for you in this guide. Here is a screenshot of what it will look like on your nodes when ran. It must be executed via root.

```
root@DS-WRK-3:~# curl --insecure -fL https://rancher-ds.[redacted]/system-agent-install.sh | sudo  sh -s - --s
erver https://rancher-ds.[redacted] --label 'cattle.io/os=linux' --token wbct666c8xhq6zc48pflwkjrfhl8xj98knm49
jf2rgqps9ksgmjbcs --ca-checksum 8976271af84fbac775f1b6cb74080b7cf1e91d3556c609737dd00b2a073ee6ab --worker
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 27723    0 27723    0     0  1592k      0 --:--:-- --:--:-- --:--:-- 1592k
[INFO]  Label: cattle.io/os=linux
[INFO]  Role requested: worker
[INFO]  Using default agent configuration directory /etc/rancher/agent
[INFO]  Using default agent var directory /var/lib/rancher/agent
[INFO]  Determined CA is necessary to connect to Rancher
[INFO]  Successfully downloaded CA certificate
[INFO]  Value from https://rancher-ds.[redacted]/cacerts is an x509 certificate
[INFO]  Successfully tested Rancher connection
[INFO]  Downloading rancher-system-agent from https://rancher-ds.[redacted]/assets/rancher-system-agent-amd64
[INFO]  Successfully downloaded the rancher-system-agent binary.
[INFO]  Generating Cattle ID
[INFO]  Successfully downloaded Rancher connection information
[INFO]  systemd: Creating service file
[INFO]  Creating environment file /etc/systemd/system/rancher-system-agent.env
[INFO]  Enabling rancher-system-agent.service
Created symlink /etc/systemd/system/multi-user.target.wants/rancher-system-agent.service → /etc/systemd/system/rancher-s
ystem-agent.service.
[INFO]  Starting/restarting rancher-system-agent.service
root@DS-WRK-3:~#
```

If everything executes correctly on the 3 nodes (the same command is used across all nodes in the same cluster), you will see a screen like this. The Ready may be false for a few minutes while everything comes up.

Condition	Status	Updated	Message
AgentDeployed	True	51 secs ago	—
BackingNamespaceCreated	True	4.3 mins ago	—
Connected	True	39 secs ago	—
Created	True	31 secs ago	—
CreatorMadeOwner	True	4.3 mins ago	—
DefaultProjectCreated	True	4.3 mins ago	—
GlobalAdminsSynced	True	1 mins ago	—
InitialRolesPopulated	True	4.3 mins ago	—
NoDiskPressure	True	4.2 mins ago	—
NoMemoryPressure	True	4.2 mins ago	—
Provisioned	True	4 mins ago	—
Ready	True	31 secs ago	—
Reconciling	False	31 secs ago	—
RKECluster	True	22 secs ago	—
SecretsMigrated	True	4.2 mins ago	—
Stalled	False	4.3 mins ago	—
SystemAccountCreated	True	1 mins ago	—
SystemProjectCreated	True	4.3 mins ago	—
Updated	Unknown	22 secs ago	[Waiting] configuring etcd node(s) custom-7eef53b96e79,custom-983569f3d4d3
Waiting	True	31 secs ago	—

After a few minutes you should see this:

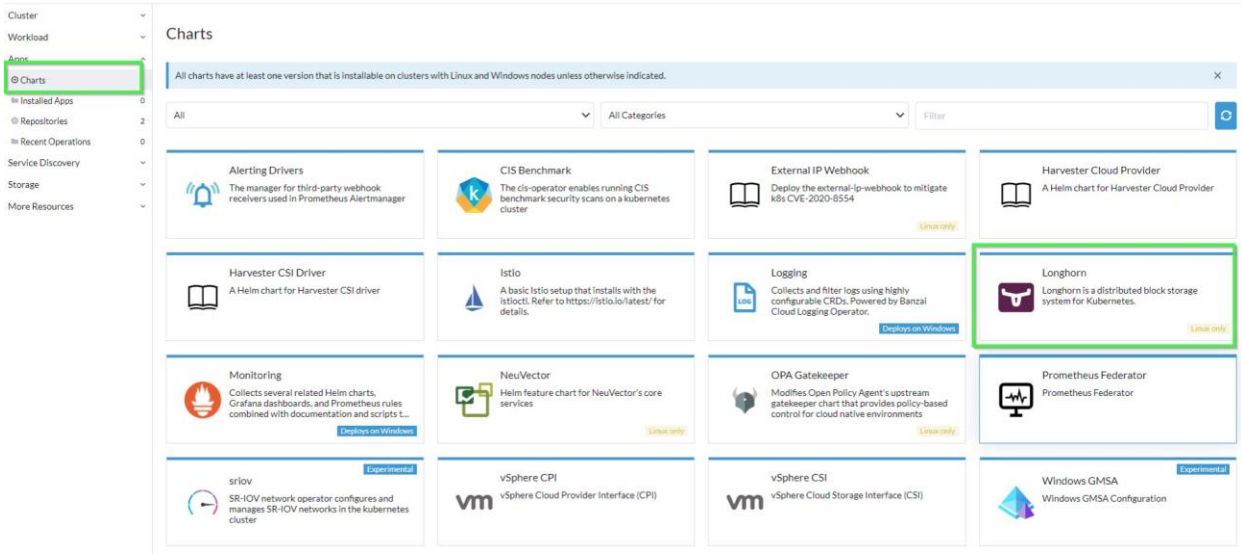
State	Name	Node	OS	Roles	Age
Running	custom-7eef53b96e79	ds-wrk-2	Linux	All	6 mins
Running	custom-983569f3d4d3	ds-wrk-3	Linux	All	6 mins
Running	custom-f697b262a1f5	ds-wrk-1	Linux	All	6 mins

At this point the worker cluster is READY! It's exciting for sure. Now we can deploy container native persistent storage. After that, it's all fun and games as we configure monitoring and deploy this workshop's version of a Hello World app: A phpBB instance with a separate mariaDB database.

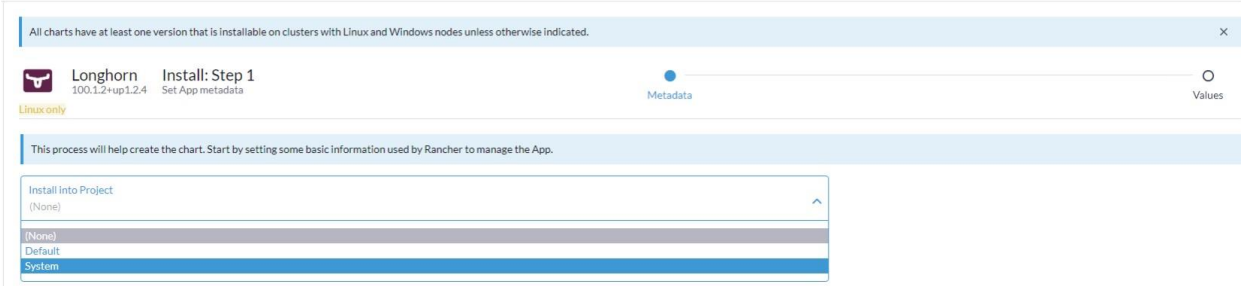
6. Deploying Longhorn for persistent storage

Longhorn provides distributed block storage as a Container Storage Interface (CSI driver) for a Kubernetes cluster. It enables applications to request and use Persistent Volumes (PV) via Persistent Volume Claims (PVC). It is easy to deploy via Rancher's built-in repository and easily upgraded via the same process. Longhorn is only supported on Linux

1. Go to Apps then Charts.
2. Ensure the Rancher repo is selected (or use All)
3. Click the Longhorn box.



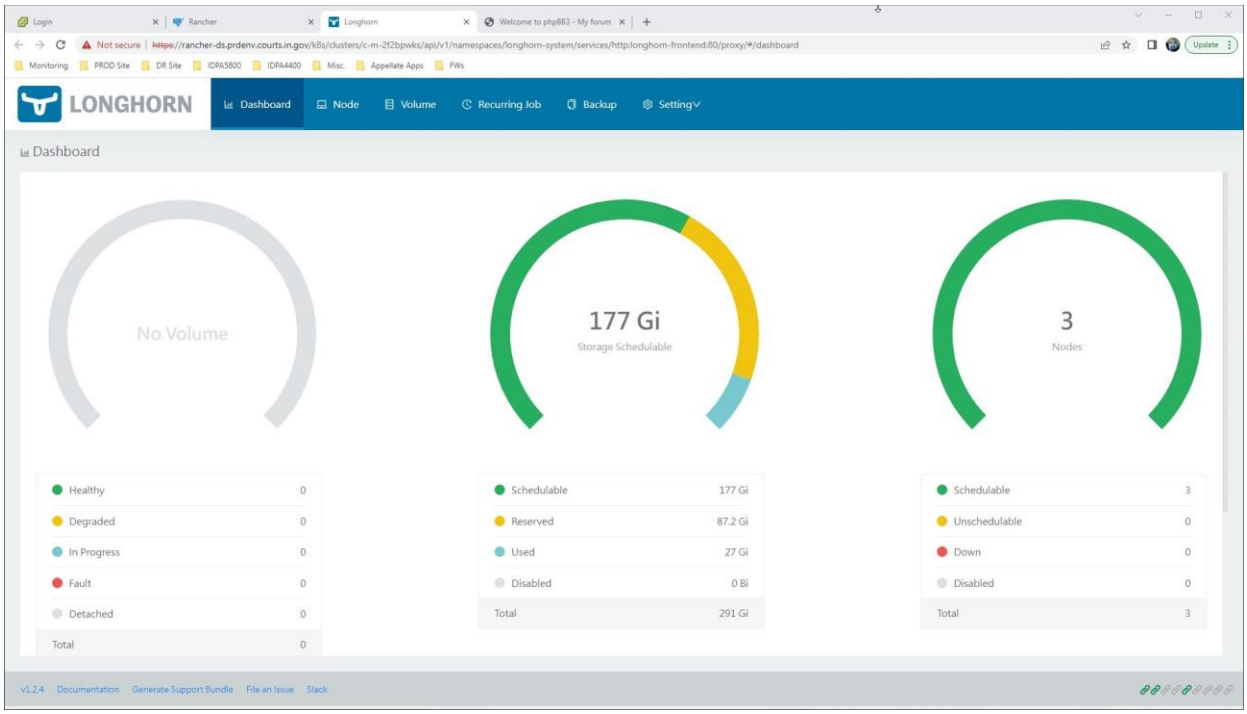
4. Feel free to read the full content of the helm chart. It's interesting.
5. Click the blue Install button.
6. Select the System project from the drop down. Click Next



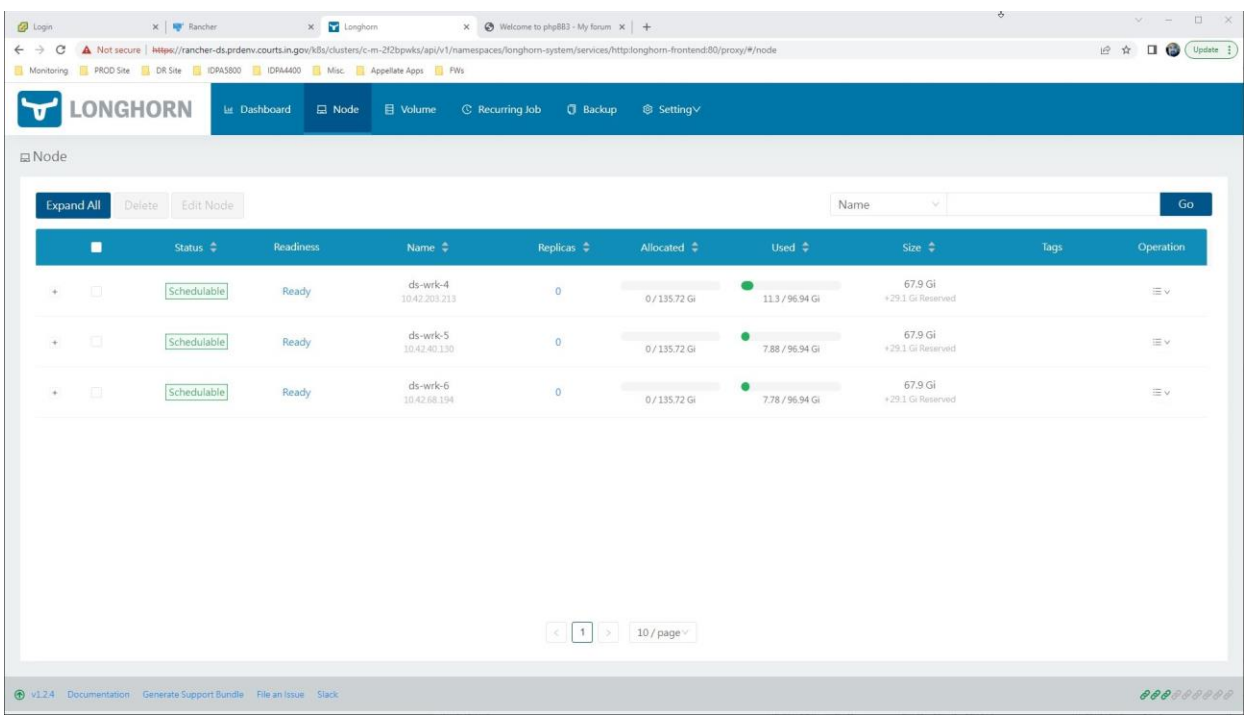
7. Everything may be left default here. Feel free to look at the options. You can set a higher resilience level if desired, but 3 pods per storage object is usually enough. Keep in mind that we should avoid using on-node storage for critical persistent workloads.
8. Click Install.
9. A console will pop up and show you the installation progress. It will be similar to this:

10. Once completed, you may click the pop-up console's X.
11. There should now be a Longhorn option in the left panel. Click that.
12. Click the Longhorn management UI button to pop it out

This is the overview dashboard:



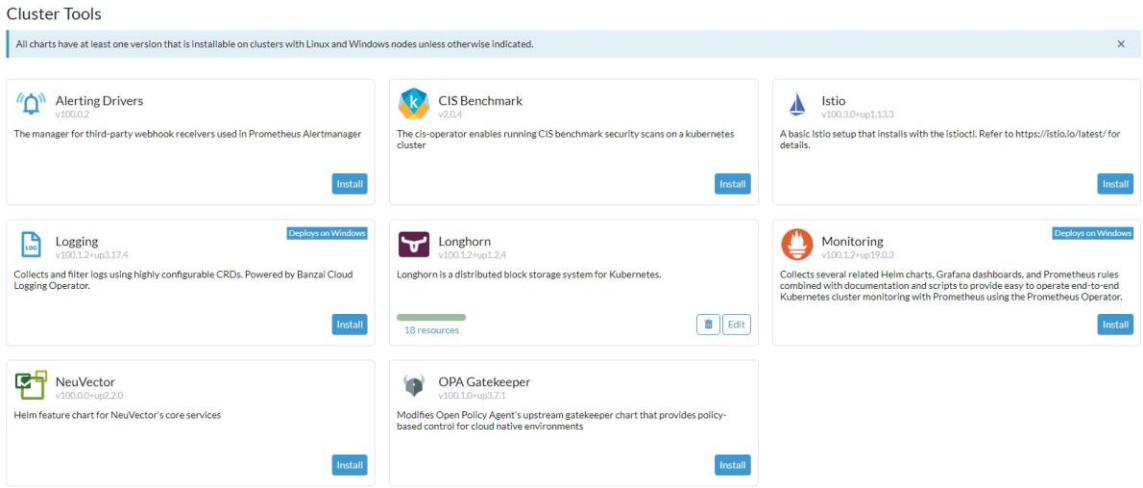
The Node tab shows stats on the 3 nodes that are associated with this Longhorn deployment. Each cluster will have its own list.



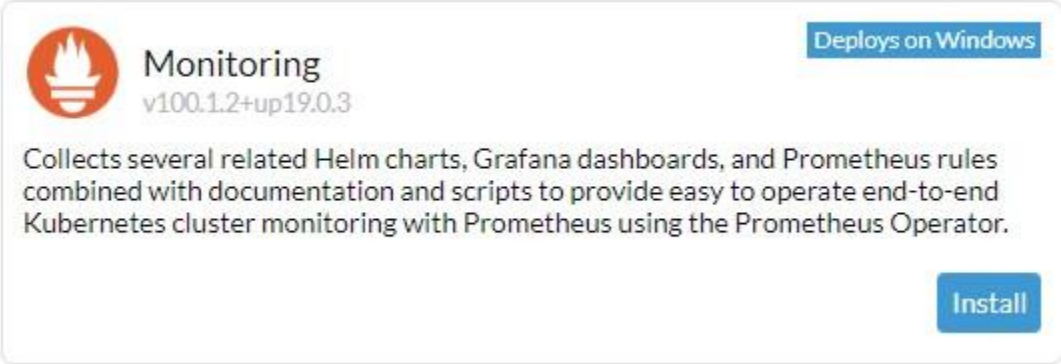
7. Deploying Monitoring for cluster metrics, performance, and stability

The monitoring infrastructure, which includes Grafana and Prometheus, can be installed via the (gear) Install Monitoring link on the Cluster Dashboard. The link will just redirect you to the Cluster tools screen where you can add cluster level features.

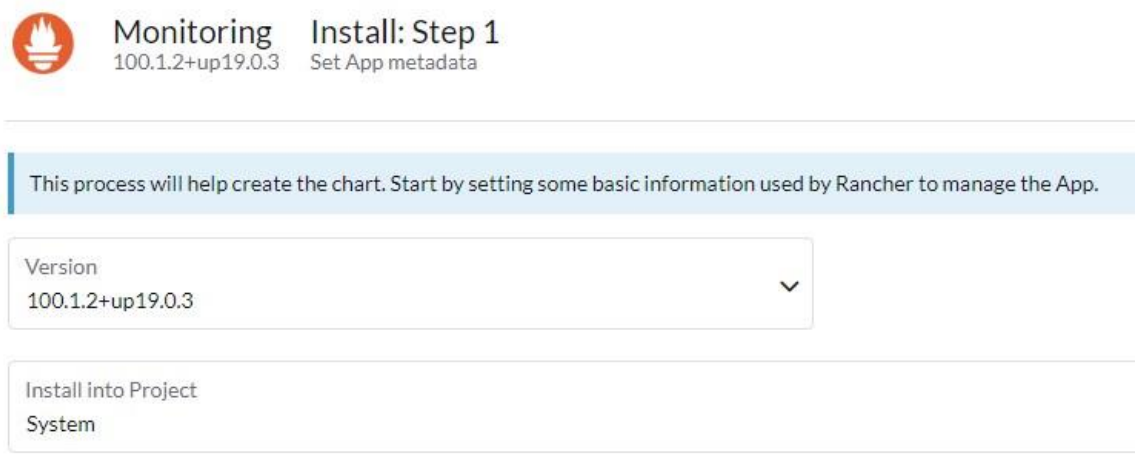
1. Click Install Monitoring.
2. The Cluster Tools screen will display:



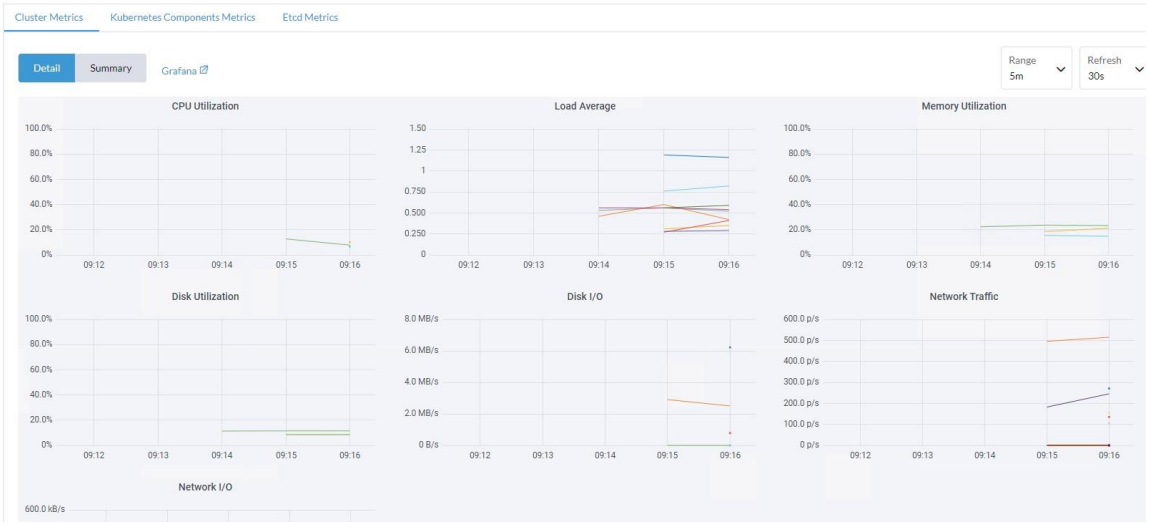
3. Click Install on the Monitoring tile:



4. Install into the System project:



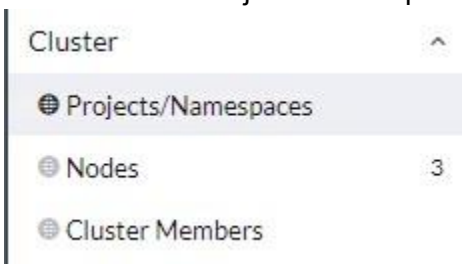
5. Click next.
6. Default values are recommended, however additional configuration may be added. It really depends on how long and persistent you need the logging data to be. Click Install.
7. A console for the deployment will display and begin scrolling as the monitoring tools are deployed. It can take a few minutes for this to deploy due to the number of pods.
8. Of note, this is integrated into the Rancher UI in some places. It is not the same as deploying a generic Helm chart.
9. Once deployed, return to the Cluster Dashboard and scroll down. You should see a new Metrics section. This has a lot of 'at a glance' information, but can also be used to launch the Grafana UI for more detailed investigation of the cluster's statistics.



8. Creating a Project and Namespace

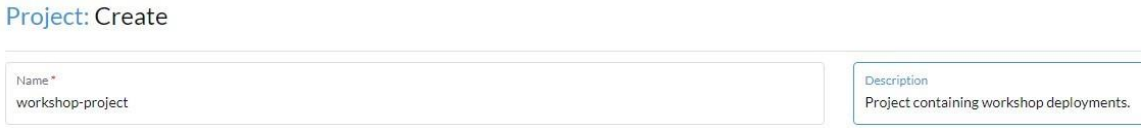
Projects in Rancher enable role-based access control for deployments and maintenance of applications, pods, etc. Namespaces are a native Kubernetes construct used to logically separate deployments, services, pods, etc. Projects can also be used to apply quotas/limits to resources.***It's critical to note*** that deleting a namespace or project will destroy underlying applications deployed within it.

1. Click Cluster > Projects/Namespaces



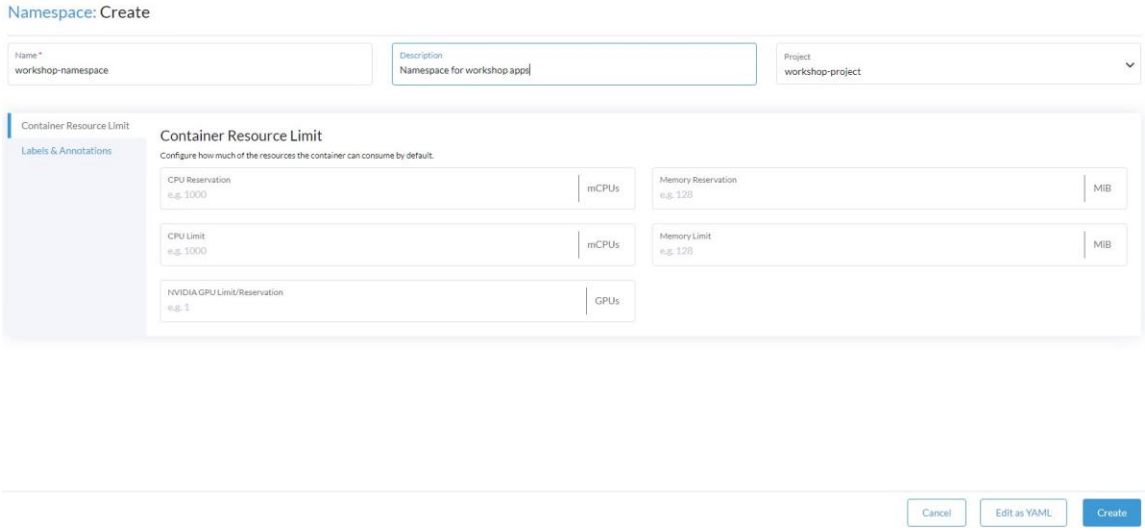
2. Click Create Project on the upper right.

3. Provide a lowercase name for the project and optionally a description.



4. Click Create.
5. Scroll down to the new workshop-project and add a Namespace.
6. Click Create Namespace

7. Provide a unique lowercase name, optionally a description, and then apply any resource reservations or limits you want to impose. Click Create.



9. Deploying our phpBB Hello World application

In this section we will add a Helm repository to our Rancher install for accessing more applications. Next we will choose to deploy an application, phpBB, which has both frontend and backend server pods with persistent storage (which will live via Longhorn). We will also customize the YAML file as part of the deployment so that we can properly access it from outside the worker cluster. Without that change it would be internal only.

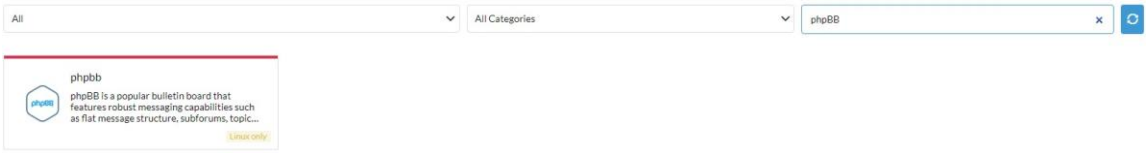
1. Open Apps in the left panel and choose repositories.

2. This view will list the currently configured container Repositories for this cluster. This is a per-cluster list. The default repositories for each cluster are Rancher and their partners with tight integration.
3. Click Create on the upper right.
4. Provide a unique name for the repository. Like everything else in Rancher/Kubernetes it must be lowercase or will give you validation errors.

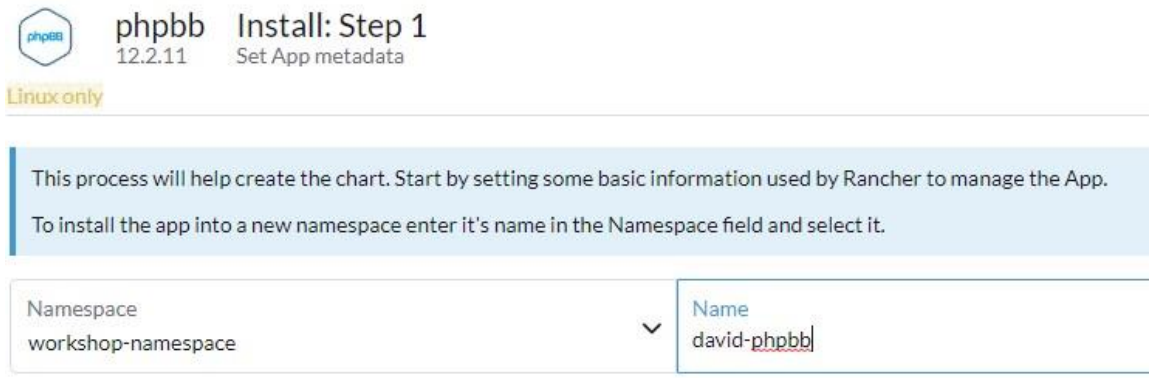
   `Bitnami repo URL: https://charts.bitnami.com/bitnami`

5. Click Create.
6. It will display In Progress while it ingests and refreshes the app inventory. It took 5 minutes for it to become Active on my Create.

7. Next we will click on Apps > Charts.
8. Click into the Filter box and type “phpBB”. There should be one result in the list.



9. Click the phpBB tile.
10. Click install in the upper right.
11. Select the workshop-namespaceNamespace and enter a unique name for your phpBB instance (lowercase). Then click Next.



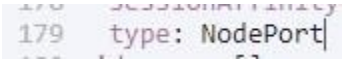
12. The next screen automatically changes to the Edit YAML user interface.
13. Scroll down to line 179

    Change:

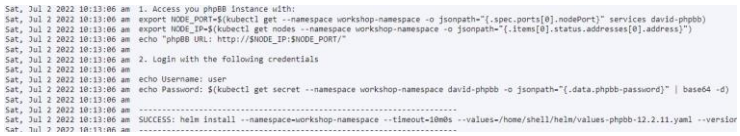
```
Type: LoadBalancer
```

    To:

```
Type: NodePort
```



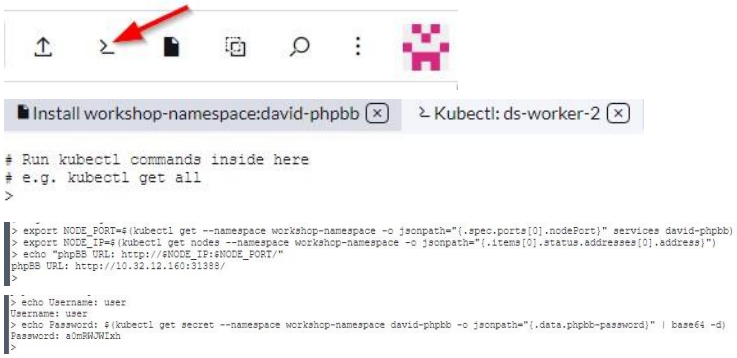
    The reason we are changing this is that we do not have Ingress or LoadBalancers configured. In Kubernetes, every node is connected in a mesh. Even if a Pod is running on Node 1, exposing the Service as a NodePort will allow a connection on Node 3 to still return responses. In this way, an external load balancer, like an F5, Citrix Netscaler, or Nginx proxy/reverse-proxy, can effectively load balance traffic across all nodes for a given port and *always* reach the expected destination.
14. Click Install.
15. It's important to see that there is some guidance at the bottom of the deployment console panel at the bottom.



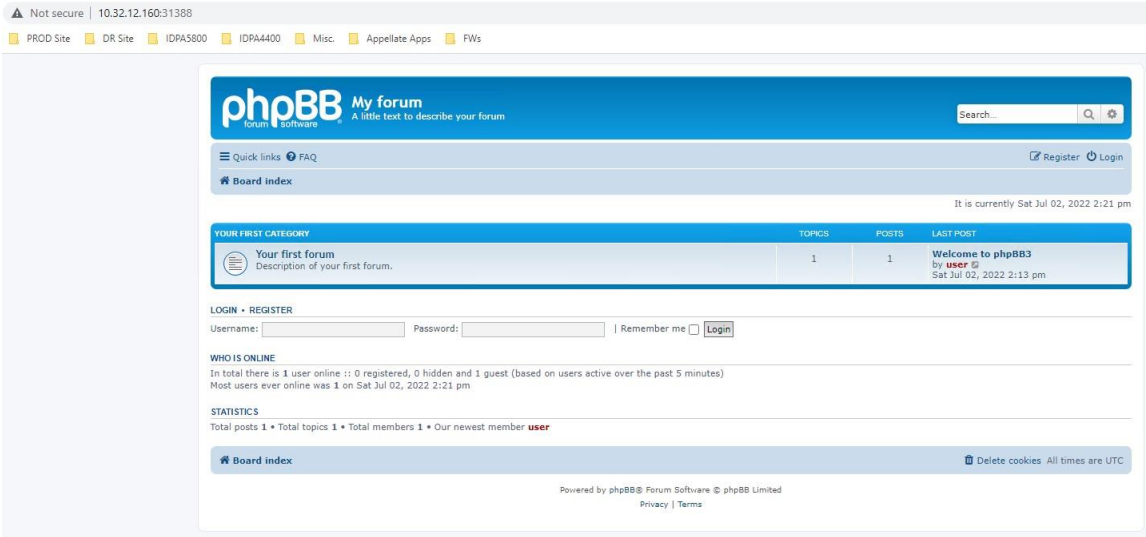
16. It explains how to get your NodePort value and admin password via the kubectl CLI. You can also find this via the user interface within Rancher. Your commands will be similar to this but not identical:

```
#1. Access you phpBB instance with:
export NODE_PORT=$(kubectl get --namespace workshop-namespace -o jsonpath="{.spec.ports[0].nodePort}" services david-phpbb) export
NODE_IP=$(kubectl get nodes --namespace workshop-namespace -o jsonpath="{.items[0].status.addresses[0].address}") echo "phpBB URL:
http://$NODE_IP:$NODE_PORT/" #2. Login with the following credentials echo Username: user
echo Password: $(kubectl get secret --namespace workshop-namespace david-phpbb -o jsonpath="{.data.phpbb-password}" | base64 -d)
```

17. To get to the command CLI from Rancher, click the terminal icon in the upper right:



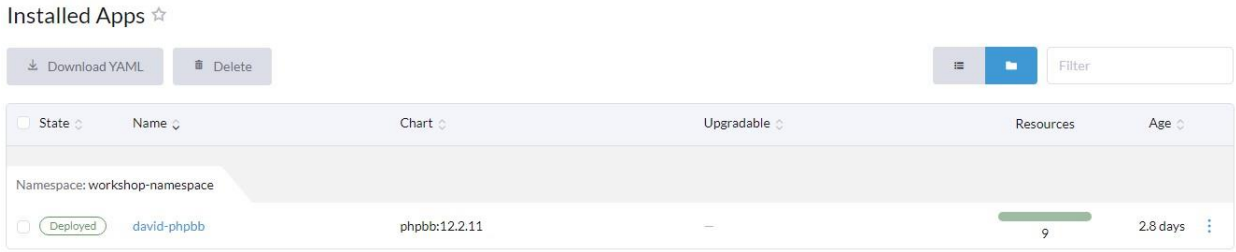
18. Now that we have our NodePort address, Administrator user name, and password combination, we can access phpBB!
19. Open a browser to your combination. Mine happens to be http://10.32.12.160:31388



20. Now you may click the login link and click through the buttons - have fun; it's yours!
21. Next we will investigate all of the components that were deployed as part of this phpBB app installation.

10. Components (Resources) of the phpBB application

Upon completing the deployment of our phpBB app, several components were created and in this section we will briefly cover them at a medium-high level. The goal is to give you a basic understanding of the component types. The descriptions below are from the Kubernetes.io documentation.



Installed App

Applications installed via the Charts screen will be displayed in the "Installed Apps" menu item. We'll click the david-phpbb link to enter that App.

Installed App: david-phpbb Deployed
Namespace: workshop-namespace Age: 2.8 days

Detail YAML

Resources Values YAML Chart README Release Notes

State	Type	Name	Namespace
Active	Secret	david-phpbb	workshop-namespace
Bound	PersistentVolumeClaim	david-phpbb	workshop-namespace
Active	Service	david-phpbb	workshop-namespace
Active	Deployment	david-phpbb	workshop-namespace
Active	ServiceAccount	david-phpbb-mariadb	workshop-namespace
Active	Secret	david-phpbb-mariadb	workshop-namespace
Active	ConfigMap	david-phpbb-mariadb	workshop-namespace
Active	Service	david-phpbb-mariadb	workshop-namespace
Active	StatefulSet	david-phpbb-mariadb	workshop-namespace

The resources list is displayed. These Resources are the components that combine to “be” your app. We’ll dive into each of these a little more.

Deployment

Deployment: david-phpbb Active
Namespace: workshop-namespace Age: 2.8 days Pod Restarts: 0

Detail Config YAML

Image: bitnami/phpbb:3.3.8-debian-11-r1 Ready: 1/1 Up-to-date: 1 Available: 1
Endpoints: 31388/TCP, 31407/TCP
Labels: app.kubernetes.io/component: phpbb app.kubernetes.io/instance: david-phpbb app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: phpbb helm.sh/chart: phpbb-12.2.11
Annotations: Show 3 annotations

Pods by State

Scale − 1 +

1
Running

Pods Metrics Conditions Related Resources

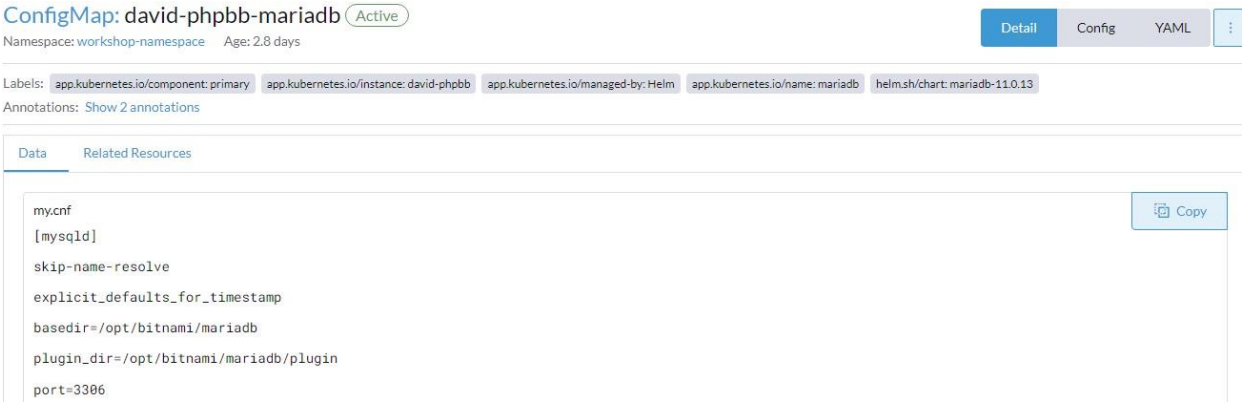
Download YAML Delete

State	Name	Node	Image	Restarts
Running	david-phpbb-7f577b495c-vdpgl	ds-wrk-4	bitnami/phpbb:3.3.8-debian-11-r1	0

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.

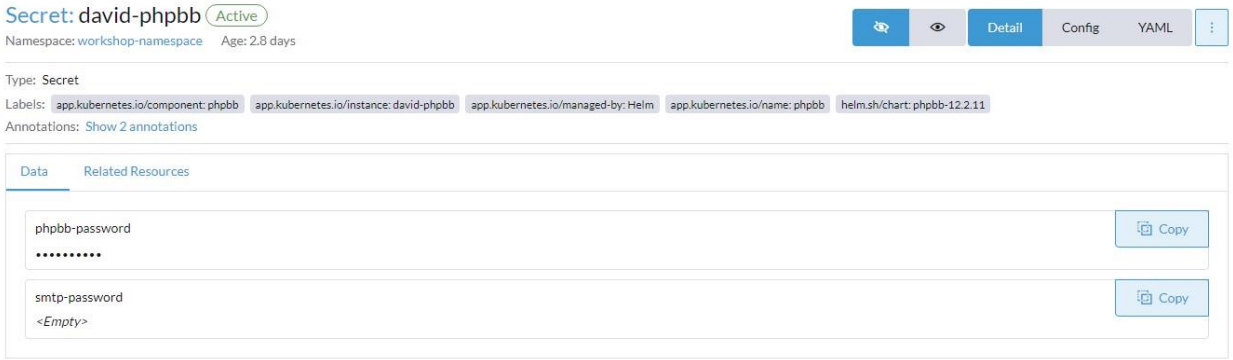
ConfigMap



A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

CAUTION: ConfigMap does not provide secrecy or encryption. If the data you want to store are confidential, use a Secret rather than a ConfigMap, or use additional (third party) tools to keep your data private.

Secret



A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.

Because Secrets can be created independently of the Pods that use them, there is less risk of the Secret (and its data) being exposed during the workflow of creating, viewing, and editing Pods. Kubernetes, and applications that run in your cluster, can also take additional precautions with Secrets, such as avoiding writing secret data to nonvolatile storage.
Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

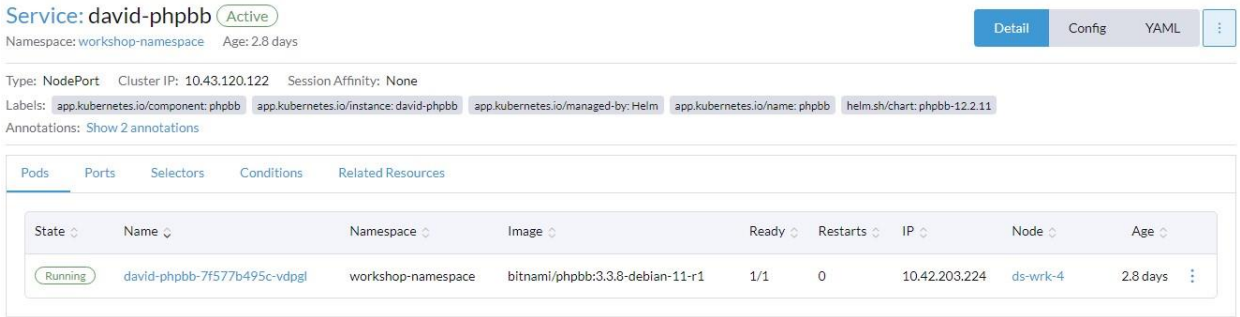
CAUTION: Kubernetes Secrets are, by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so

can anyone with access to etcd. Additionally, anyone who is authorized to create a Pod in a namespace can use that access to read any Secret in that namespace; this includes indirect access such as the ability to create a Deployment.

In order to safely use Secrets, take at least the following steps:

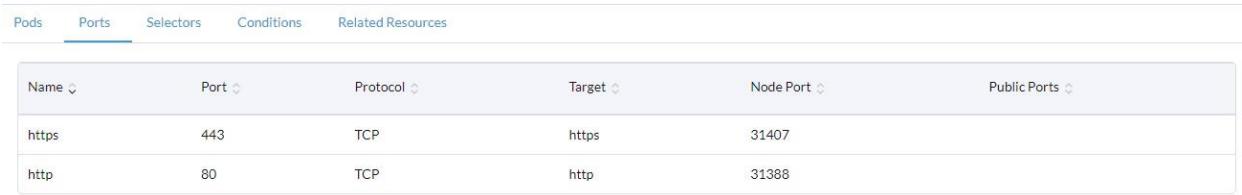
Enable Encryption at Rest for Secrets.
Enable or configure RBAC rules that restrict reading and writing the Secret. Be aware that secrets can be obtained implicitly by anyone with the permission to create a Pod.
Where appropriate, also use mechanisms such as RBAC to limit which principals are allowed to create new Secrets or replace existing ones.

Service

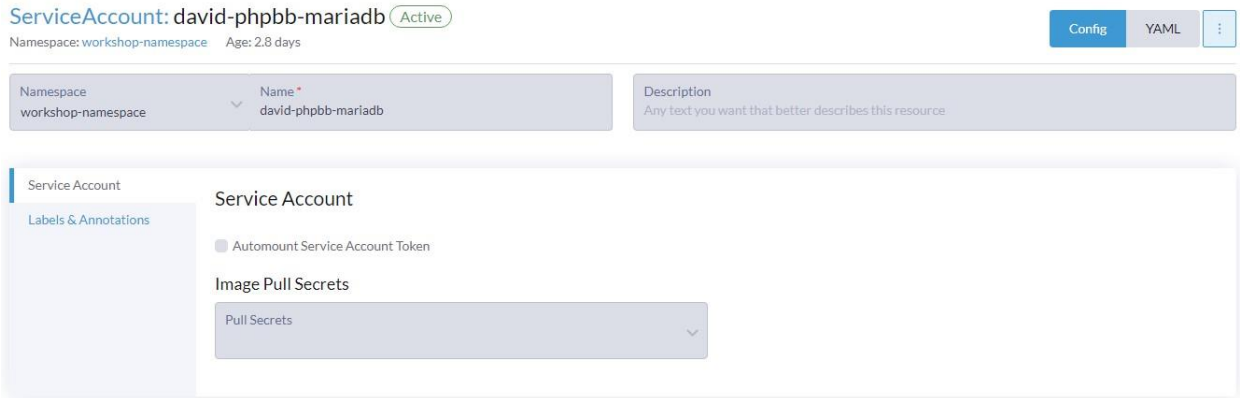


An abstract way to expose an application running on a set of Pods as a network service. With Kubernetes you don't need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them.

*This is where our customization to use the defined NodePort service came into play. By clicking the Ports tab, we can see the two ports created for this service.



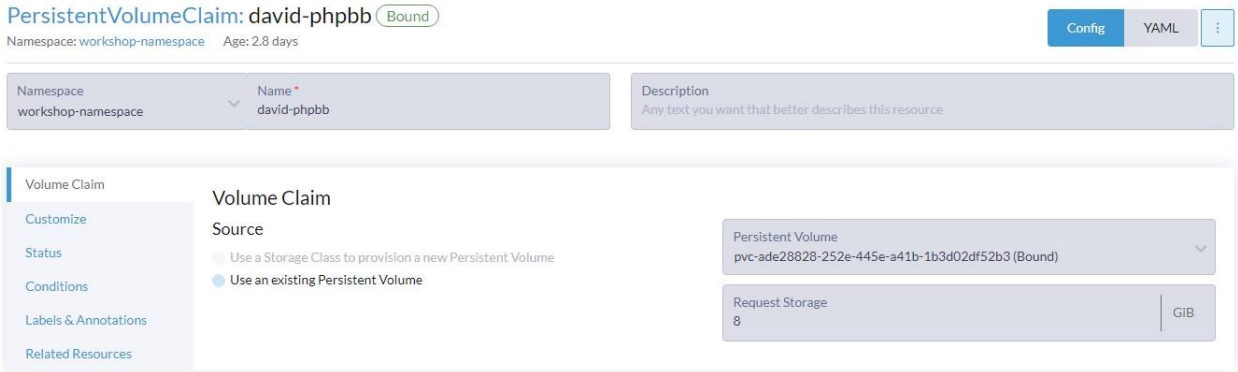
Service Account



Kubernetes distinguishes between the concept of a user account and a service account for a number of reasons:

- User accounts are for humans. Service accounts are for processes, which run in pods.
- User accounts are intended to be global. Names must be unique across all namespaces of a cluster. Service accounts are namespaced.
- Typically, a cluster's user accounts might be synced from a corporate database, where new user account creation requires special privileges and is tied to complex business processes. Service account creation is intended to be more lightweight, allowing cluster users to create service accounts for specific tasks by following the principle of least privilege.
- Auditing considerations for humans and service accounts may differ.
- A config bundle for a complex system may include definition of various service accounts for components of that system. Because service accounts can be created without many constraints and have namespaced names, such config is portable.

Persistent Volume Claim



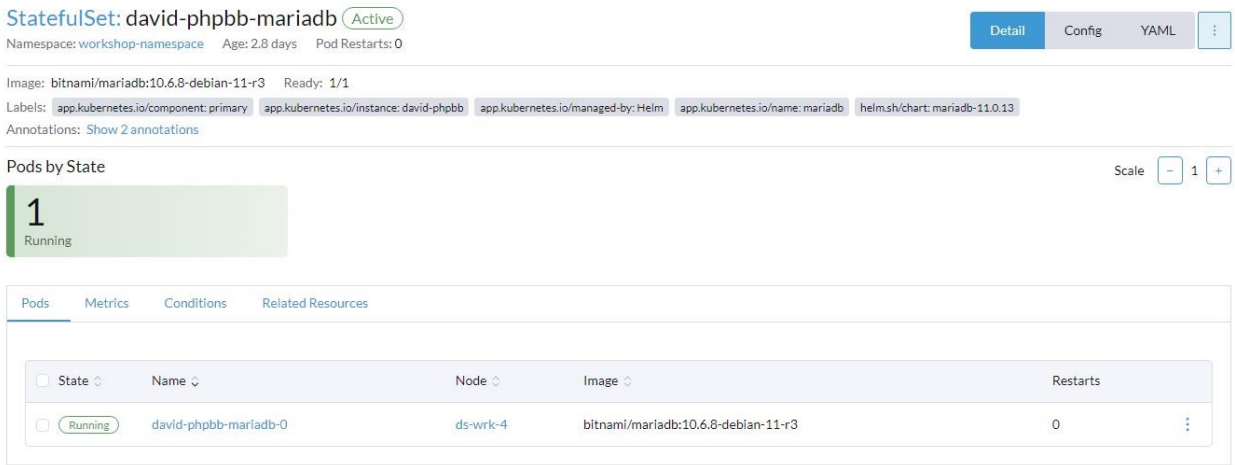
A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle

independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes).

While PersistentVolumeClaims allow a user to consume abstract storage resources, it is common that users need PersistentVolumes with varying properties, such as performance, for different problems. Cluster administrators need to be able to offer a variety of PersistentVolumes that differ in more ways than size and access modes, without exposing users to the details of how those volumes are implemented. For these needs, there is the StorageClass resource.

Statefut Sets



StatefulSet is the workload API object used to manage stateful applications.

Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

If you want to use storage volumes to provide persistence for your workload, you can use a StatefulSet as part of the solution. Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed.

Limitations

- The storage for a given Pod must either be provisioned by a PersistentVolume Provisioner based on the requested `storage class`, or pre-provisioned by an admin.
- Deleting and/or scaling a StatefulSet down will *not* delete the volumes associated with the StatefulSet. This is done to ensure data safety, which is generally more valuable than an automatic purge of all related StatefulSet resources.
- StatefulSets currently require a Headless Service to be responsible for the network identity of the Pods. You are responsible for creating this Service.
- StatefulSets do not provide any guarantees on the termination of pods when a StatefulSet is deleted. To achieve ordered and graceful termination of the pods in the StatefulSet, it is possible to scale the StatefulSet down to 0 prior to deletion. ● When using Rolling Updates with the default Pod Management Policy (`OrderedReady`), it's possible to get into a broken state that requires manual intervention to repair.