PURESTORAGE®

RANCHER®

TECHNICAL GUIDE

Rancher and Pure Service Orchestrator™

Simplify stateful application and disaggregate storage implementations.

Contents

Introduction

As modern applications are increasingly modularized and consumed as microservices, containers are a common unit of deployment. But microservices running in naked containers face management, security, and resource-allocation challenges. An orchestrator provides the necessary container runtime to power microservices while delivering the ability to abstract underlying hardware resources—the "platform"—which include CPU, memory, network, and the operating system. The orchestrator also provides other services to the container units running in a cluster, including auto-scaling, rolling upgrades, monitoring, load-balancing, and more.

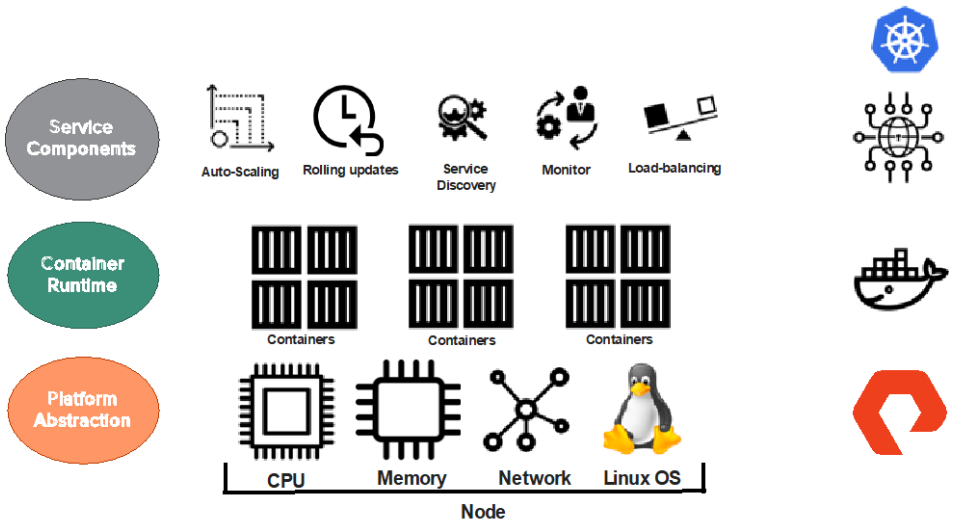

Figure 1. Container vs. orchestrator

While Docker is one of the most frequently used container runtimes in production, open-source Kubernetes is the most common among the container orchestrators. Various Kubernetes distributions are available to customers from providers like AWS, Azure, and Google, and vendors like Red Hat OpenShift, Mesosphere, Docker DataCenter, and Rancher.

Containers are historically ephemeral: Whenever you destroy a container, you lose the data associated with the application in that container. While this behavior has many use cases, there are other scenarios in which applications running in containers require persistent storage, including the need to:

- Survive restarts and outages.
- Store keys, credentials, passwords, and confidential configurations for databases.
- Access and re-use source code/binary repository data by applications running in containers like Docker.
- Provide high availability and scalability to applications running in containers.

Kubernetes allows an external provisioner called Pure Service Orchestrator™ to provision persistent storage on Pure Storage® solutions. The external provisioner runs in a pod in the Kubernetes cluster and uses a CSI driver to

mount/unmount and attach/detach persistent volumes. The CSI driver runs on all nodes that are part of the Kubernetes cluster. The provisioner is responsible for identifying the storage class requested by the Persistent Volume Claim (PVC) and provisioning the persistent volume (PV) for the corresponding PVC. Pure Service Orchestrator is a first-class Kubernetes citizen.

Objectives

The goal of this paper is to provide a simple approach to create a production-quality Kubernetes cluster using Rancher, which is open source. This document also outlines the one-time setup of pure Service Orchestrator from the Rancher catalog. PSO is an external persistent-storage provisioner from Pure Storage that runs in the Kubernetes cluster and deploys stateful application workloads in a pod with persistent storage provisioned on Pure Storage FlashArray™, FlashBlade®, and Pure Cloud Block Store™ products.

Anatomy of a Kubernetes Cluster

As an orchestrator, Kubernetes can deploy a multi-container application that would include a database, caching server, etc. Kubernetes keeps the containers running in sync for the entire application and enables scaling (up or down) an application with proper load balancing and optimum use of compute and network resources. Kubernetes also takes care of how containers communicate, handles service discovery, and provides persistent storage.

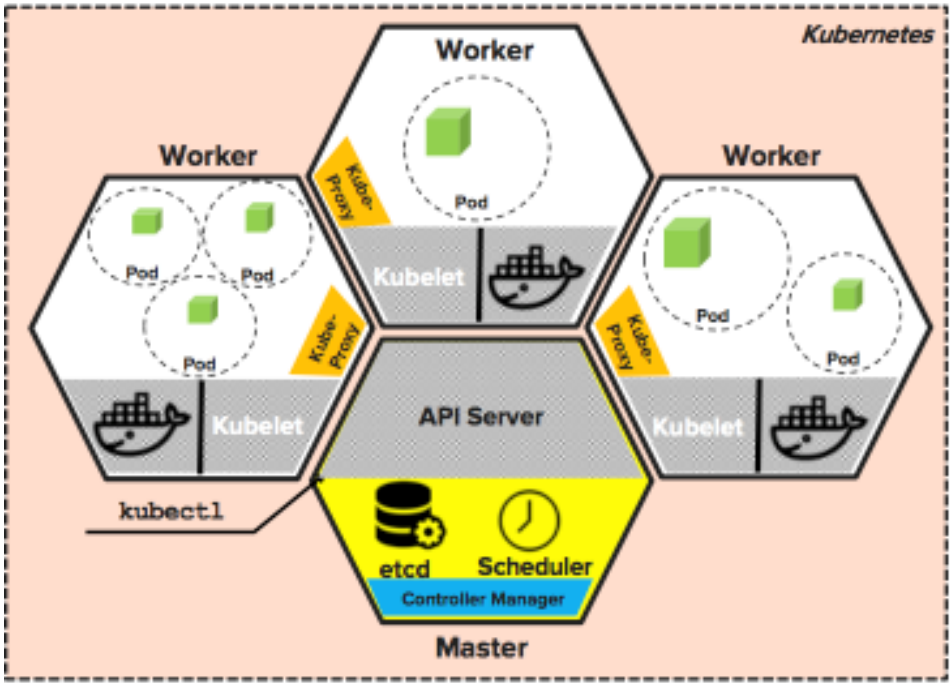

Figure 2. Four-node Kubernetes cluster

A typical Kubernetes cluster consists of a master(s) and worker nodes. Figure 2 shows the different components of a four-node Kubernetes cluster. Every Kubernetes cluster includes the control plane and runtime components. The control plane that runs in the master node primarily consists of the following components:

- **API server:** This management hub exposes the Kubernetes API to various clients, such as kubectl (the Kubernetes CLI tool), kubelet (worker component), kube-scheduler, kube-proxy, and kube-controller-manager. These APIs are responsible for scheduling, managing, and scaling different workloads and containers on different nodes in the cluster.
- **Etcd:** A key-value pair database that stores the cluster configuration data required for service discovery and deployment of distributed containerized applications. The API Server connects to etcd to access the cluster's configuration data.
- **Scheduler:** The scheduler decides where to run the created pods in the cluster.
- **Controller Manager:** The manager ensures the desired state of the workload, application, nodes (running or crashed), etc. It also determines the pod and service IP addresses in the cluster.

The runtime components consist of the following:

- **Pods:** A basic Kubernetes object that can be created, managed, and destroyed during the lifetime of an application, a pod can consist of one or many containers along with storage.
- **Service:** A naked container in a pod is ephemeral when the pod is destroyed. However, an application is supposed to be persistent in nature. Kubernetes provides an abstraction called service that groups a set of pods that are accessible over a network. While pods are the back end that may die or be recreated, services are the front-end process that is always alive and provides persistence during the life of the application.
- **Kubelet:** An agent that runs the pods in all of the worker nodes.
- **Kube-proxy:** This proxy runs in all the worker nodes to facilitate load balancing and connection forwarding to web browsers to access services in a cluster.
- **Container runtime:** Runtime provides abstracted and managed software resources like Docker containers that are needed for program execution and operation. Docker is installed on all worker nodes.

Simplifying a Kubernetes Implementation in Production

Kubernetes is becoming an industry standard for distributed application deployment. It enables robust, extensible, portable, and easy-to-migrate applications within Kubernetes environments both on-premises and in the cloud. However, this flexibility can lead to complexity, and IT professionals often lack the proper skill set to set up and configure Kubernetes clusters in production environments efficiently. In fact, 75% of respondents report inhibitions to Kubernetes adoption due to the complexity of implementing and maintaining production-grade Kubernetes clusters (Figure 3).

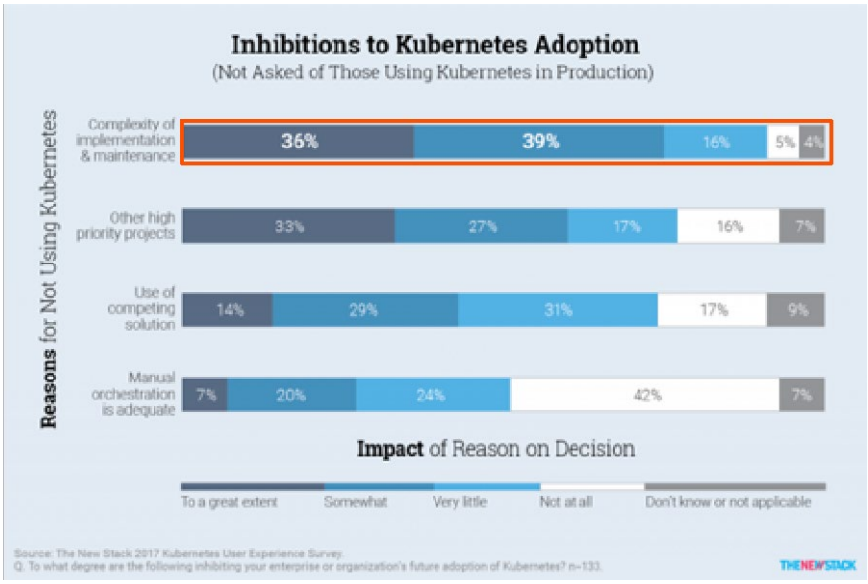

Figure 3. Top challenges with Kubernetes implementation

There are various ways to install and configure Kubernetes using Minicube, Kubeadm, and others. However, these clusters may not qualify for use in production, as you cannot easily configure more than one master during the installation process for redundancy. Further challenges arise when a container runtime like Docker and add-ons like a container network interface (CNI), helm, and other network settings are configured separately, without knowing the proper sequence of operations.

Adding and removing nodes in a Kubernetes cluster requires thoughtful planning. These manual steps all add to the overall complexity of implementing and maintaining the cluster in production.

In the light of these challenges, Rancher Kubernetes Engine (RKE)—among many other Kubernetes installers and distributions available in the market—seems to be a great solution for simplifying the Kubernetes installation process. RKE can stand up Kubernetes clusters on virtual machines and bare-metal servers.

Rancher for Kubernetes Cluster

The objective is to simplify the Kubernetes installation and configuration process for production environments. For this document, we chose to showcase the use of Rancher to install a Kubernetes cluster.

The current version of Rancher supports all flavors of Linux, including RHEL7/CentOS7 and Ubuntu 16.04. RKE serves as a core component of Rancher and provides complete lifecycle management of a Kubernetes cluster: installing, adding/removing nodes from the cluster, upgrading to new Kubernetes versions, and monitoring the health of the cluster.

Figure 4. RKE capabilities

Rancher also provides unified cluster management for multiple Kubernetes clusters hosted on-premises and in AWS, Azure, and GCP public-cloud environments (Figure 4). Rancher's unified management layer provides centralized access-control policies built on top of centralized authentication. This allows IT administrators to configure and enforce access control and security to multiple Kubernetes clusters.

Rancher's topmost layer, application workload management, provides a graphical user interface to simplify the use of containers for applications listed in the Catalog. You can add custom applications that are not present in the Rancher Catalog using the Helm chart repository for one-click deployments.

At the time of writing, the most recent version of Rancher 2.1.7 and RKE version 0.1.7 along with Kubernetes v1.13.4 and docker-ce-18.09.2 for CentOS 7.5 was used for this validation. This document will highlight some of the key requirements for creating a Kubernetes cluster on CentOS7.5 clients. For more details, refer to the official Rancher documentation.

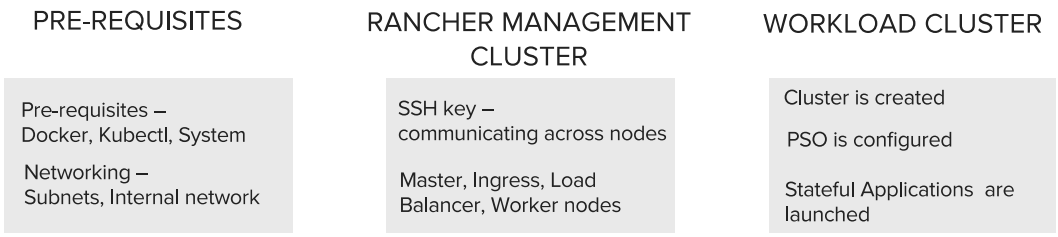

Figure 5. Kubernetes cluster creation.

Administrators can handle Kubernetes cluster creation in three phases—pre-requisites, setup Rancher management cluster, followed by workload cluster.

Phase 1: Prerequisites

The planning phase includes the most important part—network setup. Apart from public IP addresses assigned to the nodes within the Kubernetes cluster, a few network settings are required for a basic Kubernetes cluster (see Figure 6). The IP addresses listed are examples for the purposes of this paper.

- An internal IP network on which the cluster nodes can communicate among themselves. This could be a private network like 192.168.x.x.

- There should also be classless interdomain routing (CIDR) ranges—pod_cidr_range for the pod network, something like 172.16.0.0/16 and service_cidr_range for the service network, and something like 172.20.0.0/16, respectively. These IP ranges should not be used elsewhere in the network.

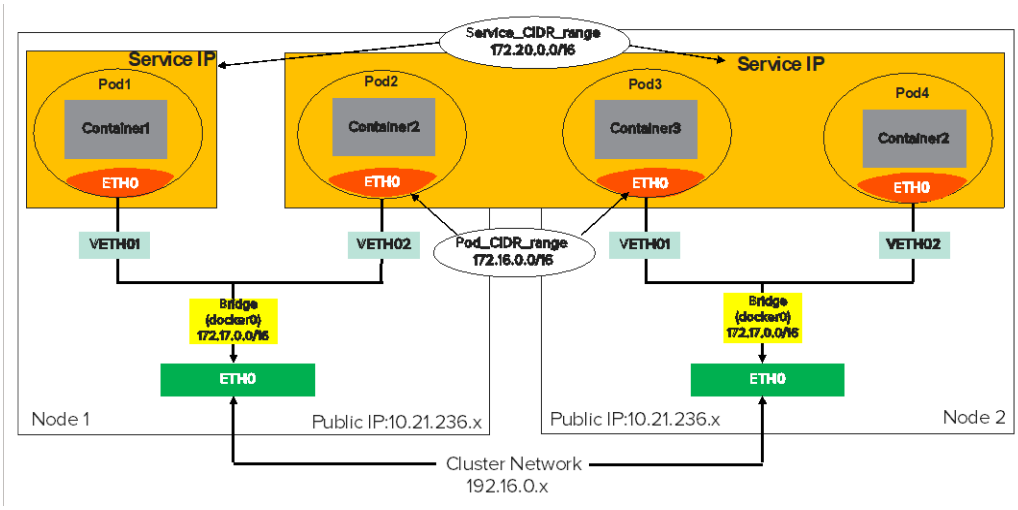

Figure 6. Basic network layout for services, pods, and nodes

Setting up Environment Variables on Linux Nodes for Kubernetes and Docker Components

The following steps list how you can install and configure docker-ce (community edition) on each of the nodes and the rest of the Linux nodes that would perform Docker push/pull operations to the private Docker Registry.

Disable SELinux on all Linux nodes.

```
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
```

Enable the bridge for the Docker—br_netfilter kernel module.

```
modprobe br_netfilter
echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
```

Disable swap on all Linux nodes on which Docker is configured.

```
swapoff -a
```

Comment out the swap entry in /etc/fstab to make the change persistent across reboots.

```
cat /etc/fstab |grep swap
#/dev/mapper/vg0-lv_swap swap                    defaults        0 0
```

Install docker 18.09.2 on all the nodes that are part of the Rancher and workload clusters. This version Docker provides the Docker Security Update:CVE-2019-5736.

```
[root@sn1-r720-g09-15 ~]# curl https://releases.rancher.com/install-docker/18.09.2.sh | sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 15225  100 15225    0     0  86477      0 --:--:-- --:--:-- --:--:-- 87000
+ '[' centos = redhat ']'
+ sh -c 'yum install -y -q yum-utils'
Package yum-utils-1.1.31-50.el7.noarch already installed and latest version
+ sh -c 'yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo'
Loaded plugins: fastestmirror, langpacks
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to /etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
+ '[' stable '!=' stable ']'
+ sh -c 'yum makecache fast'
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
epel/x86_64/metalink                                                    |  14 kB  00:00:00
 * base: mirror.keystealth.org
 * epel: mirror.prgmr.com
 * extras: mirror.dal10.us.leaseweb.net
 * ius: mirrors.kernel.org
 * updates: centos-distro.cavecreek.net
base                                                                    | 3.6 kB  00:00:00
docker-ce-stable                                                        | 3.5 kB  00:00:00
extras                                                                  | 3.4 kB  00:00:00
ius                                                                     | 2.3 kB  00:00:00
logstash-6.x                                                            | 1.3 kB  00:00:00
puretec                                                                 | 2.9 kB  00:00:00
updates                                                                 | 3.4 kB  00:00:00
Metadata Cache Created
+ sh -c 'yum install -y -q docker-ce-18.09.2'
+ '[' -d /run/systemd/system ']'
+ sh -c 'service docker start'
Redirecting to /bin/systemctl start docker.service
+ sh -c 'docker version'
Client:
 Version:           18.09.3
 API version:       1.39
 Go version:        go1.10.8
 Git commit:        774a1f4
 Built:             Thu Feb 28 06:33:21 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.2
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.6
  Git commit:       6247962
  Built:            Sun Feb 10 03:47:25 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

If you would like to use Docker as a non-root user, you should now consider adding your user to the "docker" group with something like:

```
sudo usermod -aG docker your-user
```

Remember that you will have to log out and back in for this to take effect!

Warning: Adding a user to the "docker" group will grant the ability to run containers which can be used to obtain root privileges on the docker host.

Refer to https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface for more information.

Install **Kubectl** on the Rancher management cluster nodes.

```
yum install -y kubectl
```

After the nodes come back up, perform the following steps.

```
systemctl start docker && systemctl enable docker
systemctl start kubelet && systemctl enable kubelet
```

Create user **-rke** with no password privileges.

```
[root@sn1-r720-g09-15 ~]# useradd -d /home/rke -m rke
[root@sn1-r720-g09-17 ~]# passwd rke
Changing password for user rke.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

   echo "rke ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/rke
chmod 0440 /etc/sudoers.d/rke
usermod -aG docker rke
```

Edit the /etc/docker/daemon.json on all the Rancher nodes to include the following lines and **restart** docker.

```
     vi /etc/docker/daemon.json
{
  "group": "docker",
  "storage-driver": "overlay2",
  "storage-opts": [
     "overlay2.override_kernel_check=true"
  ]
}
```

As a best practice, we recommend having a minimum of three nodes (VMs or bare-metal) for creating the Rancher management cluster. All three nodes can run etcd, control, and worker nodes for the cluster. Each physical or virtual node should have at least 8 CPU cores, 32GB memory, and 50GB to 200GB storage in production environments. For more information, refer to the sizing documentation. In addition to the three nodes for Rancher management cluster, you need another node to configure the external load-balancer like nginx and install Rancher and rke binaries. Figure 7 illustrates the schematic diagram of the Rancher management cluster used in this validation.

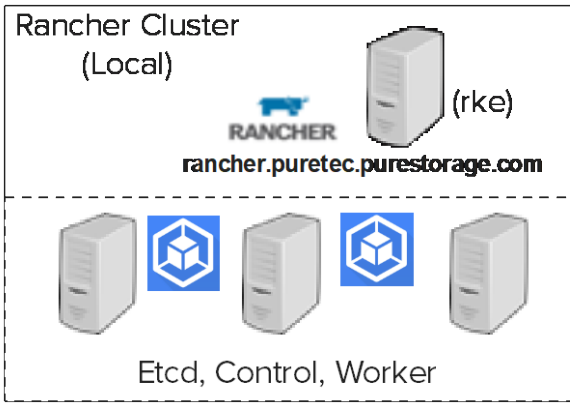

Figure 7. Three-node Rancher Management cluster with RKE running on a separate node

Make sure to include an entry in the DNS to resolve "rancher.puretec.purestorage.com" to its corresponding IP address. In this example rancher.puretec.purestorage.com resolves to 10.21.236.114 (the installer node).

```
            [rke@sn1-r720-g09-15 ~]$ nslookup rancher.puretec.purestorage.com
Server:             10.21.93.16
Address:    10.21.93.16#53

Name:       rancher.puretec.purestorage.com
Address: 10.21.236.114
```

Configure nginx on installer node where rke binaries will be installed.

```
yum install epel-release
yum install nginx
systemctl start nginx
```

Add the three Rancher nodes to the /etc/nginx/nginx.conf file for "nginx" to function as the load-balancer and listen on port 80. Make sure to add **line 1** to the **nginx.conf** file for CentOS clients.

```
cat nginx.conf
load_module '/usr/lib64/nginx/modules/ngx_stream_module.so';

worker_processes 4;
worker_rlimit_nofile 40000;

events {
    worker_connections 8192;
}

http {
    server {
        listen         80;
        return 301 https://$host$request_uri;
    }
}

stream {
    upstream rancher_servers {
        least_conn;
        server 10.21.236.115:443 max_fails=3 fail_timeout=5s;
        server 10.21.236.116:443 max_fails=3 fail_timeout=5s;
        server 10.21.236.117:443 max_fails=3 fail_timeout=5s;
    }
```

```
        server {
            listen     443;
            proxy_pass rancher_servers;
        }
    }
```

Once the /etc/nginx/nginx.conf is configured with the right set of Linux nodes used for the Rancher management cluster, nginx should be reloaded.

```
nginx -s reload
```

Verify if nginx is running with the new configuration.

```
   netstat -nlp|grep nginx
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      27319/nginx: master
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      27319/nginx: master

netstat -nlp|grep 80
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      27319/nginx: master
```

Phase 2: Rancher Management Cluster

The following steps validate all the prerequisites before creating a Kubernetes cluster. Create a user "rke" with no password apart from prerequisites for networking, Docker engine, and other Linux node resource tuning, then login as **rke** user and download the latest GA version of rke_linux_amd64.

```
[rke@sn1-r720-g09-15 ~]$ pwd
/home/rke
[rke@sn1-r720-g09-15 ~]$ ls -al rke*
-rw-r--r-- 1 rke rke 32561840 Jan  8 11:53 rke_linux-amd64

[rke@sn1-r720-g09-15 ~]$ mv rke_linux-amd64 rke

[rke@sn1-r720-g09-15 ~]$ chmod +x rke
[rke@sn1-r720-g09-15 ~]$ ./rke --version
rke version v0.1.17
```

As "rke" user, generate the private ssh key on the installer node and copy to all the nodes that are part of the Rancher management cluster.

```
[rke@sn1-r720-g09-15 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/rke/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/rke/.ssh/id_rsa.
Your public key has been saved in /home/rke/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:UBTzJLJ8Mw5I46jUQ0gm4zTxYhmIKXzvq0JDnzYq4mA rke@sn1-r720-g09-15.puretec.purestorage.com
The key's randomart image is:
+---[RSA 2048]----+
|BOo.o ..*..      |
|X+B= + + =       |
|.*o++ = + .      |
|oo. .. = o       |
|o . o   S        |
| o = .           |
|oE+ . .          |
|=o   .           |
|+....            |
```

```
+----[SHA256]-----+

[rke@sn1-r720-g09-15 ~]$ cd .ssh
[rke@sn1-r720-g09-15 .ssh]$ ls -al
total 20
drwxrwxr-x 2 rke rke 4096 Jan  9 12:18 .
drwx------ 5 rke rke 4096 Jan  9 12:17 ..
-rw------- 1 rke rke 1675 Jan  9 12:18 id_rsa
-rw-r--r-- 1 rke rke  425 Jan  9 12:18 id_rsa.pub
-rw------- 1 rke rke  191 Jan  9 12:11 known_hosts.old
```

Copy the id_rsa to all the hosts that are part of the Rancher management cluster. In the following example, the id_rsa_.pub file is copied to the installer host sn1-r720-g09-15 followed by all the remaining nodes that are going to be part of the Rancher management cluster.

```
[rke@sn1-r720-g09-15 .ssh]$ ssh-copy-id rke@sn1-r720-g09-15
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/rke/.ssh/id_rsa.pub"
The authenticity of host 'sn1-r720-g09-15 (10.21.236.114)' can't be established.
ECDSA key fingerprint is SHA256:WPfPKoz3szaycfJlevZZ+Js/DUaR4Vape0WQ47n2Avo.
ECDSA key fingerprint is MD5:e7:c1:f5:e7:c3:18:19:4d:fe:a7:d1:b6:bc:83:24:de.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already
installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the
new keys
rke@sn1-r720-g09-15's password:

Number of key(s) added: 1
```

Now try logging into the machine, with: "ssh 'rke@sn1-r720-g09-15'" and check to make sure that only the key(s) you wanted were added.

The following example illustrates the step to copy the id_rsa file to one of the hosts that is part of the Rancher management cluster. Similar steps need to be taken to copy the file on to the remaining two nodes in the Rancher management cluster.

```
[rke@sn1-r720-g09-15 .ssh]$ ssh-copy-id rke@sn1-r720-g09-17
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/rke/.ssh/id_rsa.pub"
The authenticity of host 'sn1-r720-g09-17 (10.21.236.115)' can't be established.
ECDSA key fingerprint is SHA256:spc3ksO5TcGlO0MBC5bfj/CsYzlUxLx3NmocD50ZKvI.
ECDSA key fingerprint is MD5:b9:d9:85:e1:ff:a8:53:03:37:e8:52:77:10:65:cc:ba.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already
installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the
new keys
rke@sn1-r720-g09-17's password:

Number of key(s) added: 1
```

Now try logging into the machine, with: "ssh 'rke@sn1-r720-g09-17'" and check to make sure that only the key(s) you wanted were added.

The rancher-cluster.yaml file is created with the IP address information of all the nodes that are part of the Rancher management cluster, along with their respective roles—etcd, control, and worker. It is recommended to assign all three roles to all three nodes in the Rancher management cluster. Make sure to include the path to the ssh_key_path for the id_rsa. The following is a sample rancher-cluster.yaml file.

```
[rke@sn1-r720-g09-15 ~]$ cat rancher-cluster.yml
nodes:
  - address: 10.21.236.115
    internal_address: 192.16.0.115
    user: rke
    role: [controlplane,worker,etcd]
  - address: 10.21.236.116
    internal_address: 192.16.0.116
    user: rke
    role: [controlplane,worker,etcd]
  - address: 10.21.236.117
    internal_address: 192.16.0.117
    user: rke
    role: [controlplane,worker,etcd]

ssh_key_path: ~/.ssh/id_rsa

services:
  etcd:
    snapshot: true
    creation: 6h
    retention: 24h
```

Now all the pre-requisites are complete and the install file rancher-cluster.yaml is ready. Run the rancher-cluster.yaml file using “rke” as “rke” user.

```
[rke@sn1-r720-g09-15 ~]$ ./rke up --config ./rancher-cluster.yml
```

Now the Rancher management cluster is up and running. The Helm chart needs to be installed and configured, followed by Rancher. It is recommended to install Helm 2.12 or later. Helm is a package manager for Kubernetes, using a packaging format called charts that contains a combination of one or many files that are responsible for assigning a set of resources in Kubernetes.

```
[rke@sn1-r720-g09-15 ~]$ tar xvf helm-v2.12.1-linux-amd64.tar

[rke@sn1-r720-g09-15 ~]$ kubectl -n kube-system create serviceaccount tiller

[rke@sn1-r720-g09-15 ~]$ kubectl create clusterrolebinding tiller   --clusterrole cluster-admin   --
serviceaccount=kube-system:tiller

[rke@sn1-r720-g09-15 ~]$ helm init --service-account tiller

[rke@sn1-r720-g09-15 ~]$ kubectl -n kube-system  rollout status deploy/tiller-deploy
deployment "tiller-deploy" successfully rolled out
[rke@sn1-r720-g09-15 ~]$ helm version
Client: &version.Version{SemVer:"v2.12.1", GitCommit:"02a47c7249b1fc6d8fd3b94e6b4babf9d818144e",
GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.12.1", GitCommit:"02a47c7249b1fc6d8fd3b94e6b4babf9d818144e",
GitTreeState:"clean"}
 [rke@sn1-r720-g09-15 ~]$ helm install stable/cert-manager \
>   --name cert-manager \
>   --namespace kube-system
NAME:   cert-manager
LAST DEPLOYED: Fri Jan 11 08:05:09 2019
NAMESPACE: kube-system
STATUS: DEPLOYED

RESOURCES:
==> v1beta1/Deployment
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
cert-manager  1        0        0           0          0s
```

```
==> v1/Pod(related)
NAME                            READY  STATUS   RESTARTS  AGE
cert-manager-7d4bfc44ff-gzd4t  0/1    Pending  0         0s

==> v1/ServiceAccount
NAME          SECRETS  AGE
cert-manager  1        0s

==> v1beta1/ClusterRole
NAME          AGE
cert-manager  0s

==> v1beta1/ClusterRoleBinding
NAME          AGE
cert-manager  0s


NOTES:
cert-manager has been deployed successfully!

In order to begin issuing certificates, you will need to set up a ClusterIssuer or Issuer resource (for
example, by creating a 'letsencrypt-staging' issuer).

More information on the different types of issuers and how to configure them can be found in our
documentation:
https://cert-manager.readthedocs.io/en/latest/reference/issuers.html

For information on how to configure cert-manager to automatically provision Certificates for Ingress
resources, take a look at the `ingress-shim` documentation: https://cert-manager.readthedocs.io/en/latest/reference/ingress-
shim.html

[rke@sn1-r720-g09-15 ~]$ kubectl -n kube-system rollout status deploy/cert-manager
deployment "cert-manager" successfully rolled out
```

The following steps list the process to install rancher and access “rancher.puretec.purestorage.com.”

```
[rke@sn1-r720-g09-15 ~]$ helm repo add rancher-latest https://releases.rancher.com/server-charts/latest
"rancher-latest" has been added to your repositories


[rke@sn1-r720-g09-15 ~]$ helm install rancher-latest/rancher --name rancher --namespace cattle-system --set
hostname=rancher.puretec.purestorage.com
NAME:   rancher
LAST DEPLOYED: Fri Jan 11 10:52:53 2019
NAMESPACE: cattle-system
STATUS: DEPLOYED

RESOURCES:
==> v1/ClusterRoleBinding
NAME     AGE
rancher  1s

==> v1/Service
NAME     TYPE       CLUSTER-IP   EXTERNAL-IP  PORT(S)  AGE
rancher  ClusterIP  10.43.91.78  <none>       80/TCP   1s

==> v1/Deployment
NAME     DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
rancher  3        0        0           0          1s

==> v1beta1/Ingress
NAME     HOSTS                           ADDRESS  PORTS  AGE
rancher  rancher.puretec.purestorage.com  80, 443  1s
```

```
==> v1alpha1/Issuer
NAME     AGE
rancher  1s

==> v1/Pod(related)
NAME                       READY  STATUS   RESTARTS  AGE
rancher-9c5854bb6-8bnsx    0/1    Pending  0         1s
rancher-9c5854bb6-9llst    0/1    Pending  0         1s
rancher-9c5854bb6-mlfnf    0/1    Pending  0         1s

==> v1/ServiceAccount
NAME     SECRETS  AGE
rancher  1        1s


NOTES:
Rancher Server has been installed.
NOTE: Rancher may take several minutes to fully initialize. Please standby while Certificates are being issued
and Ingress comes up.
Check out our docs at https://rancher.com/docs/rancher/v2.x/en/
Browse to https://rancher.puretec.purestorage.com


Happy Containering!
```

Include the following lines in the .bash_profile and .bashrc scripts present in the home directory of user “rke” to seamlessly use “kubectl” commands after login as user “rke,” or after a reboot.

```
[rke@sn1-r720-g09-15 ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH

export KUBECONFIG=/home/rke/kube_config_rancher-cluster.yml

export PATH=$PATH:/home/rke/linux-amd64

[rke@sn1-r720-g09-15 ~]$ cat .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

export KUBECONFIG=/home/rke/kube_config_rancher-cluster.yml
export PATH=$PATH:/home/rke/linux-amd64

# User specific aliases and functions

[rke@sn1-r720-g09-15 ~]$ kubectl get pods -n cattle-system
NAME                                    READY   STATUS    RESTARTS   AGE
```

```
cattle-cluster-agent-6586b96bc5-h8nh2   1/1     Running   0          1h
cattle-node-agent-9lmmt                 1/1     Running   0          1h
cattle-node-agent-gshdf                 1/1     Running   0          1h
cattle-node-agent-nx8fc                 1/1     Running   0          1h
rancher-9c5854bb6-8bnsx                 1/1     Running   0          1h
rancher-9c5854bb6-9llst                 1/1     Running   0          1h
rancher-9c5854bb6-mlfnf                 1/1     Running   0          1h
```

Now Rancher is up and running. You can use "https://rancher.puretec.turestorage.com" to launch the cluster.

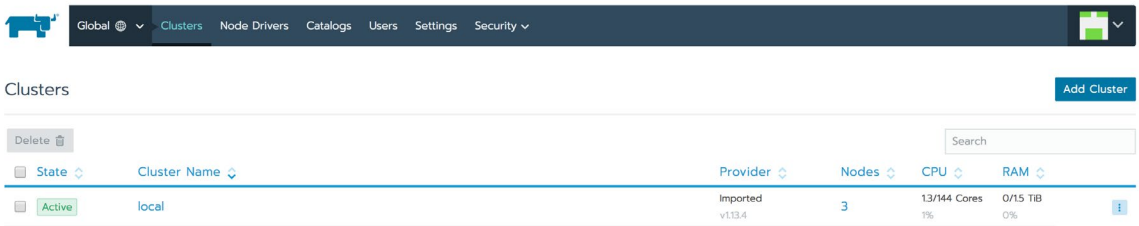

Figure 8. The "local" Rancher Management Cluster listed in the Rancher GUI

The three nodes are listed in the "local" cluster that represents the Rancher Management Cluster.

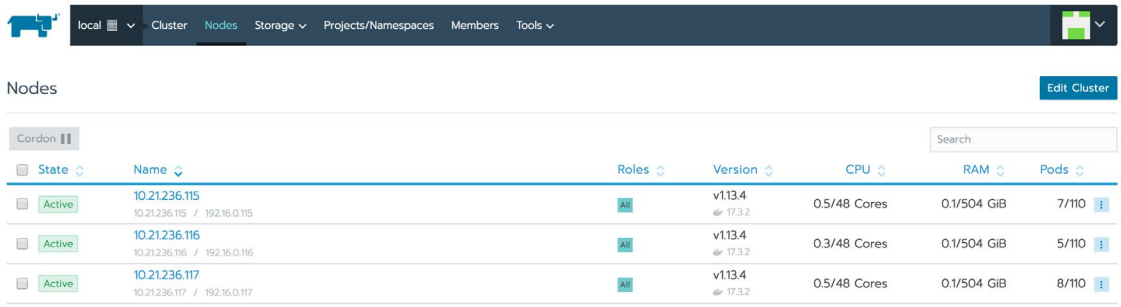

Figure 9. Three-node Rancher management cluster (recommended)

Phase 3: Workload Cluster

At the time of writing, a brand new, seven-node workload cluster called "pure-new-solutions" is added to the Rancher management cluster (local). The workload cluster is responsible for running any container-based applications in the pure-new-solutions cluster. It uses three nodes for etcd and control plane; the remaining four nodes are workers. Figure 10 illustrates the new workload cluster that is added to the existing Rancher management cluster.

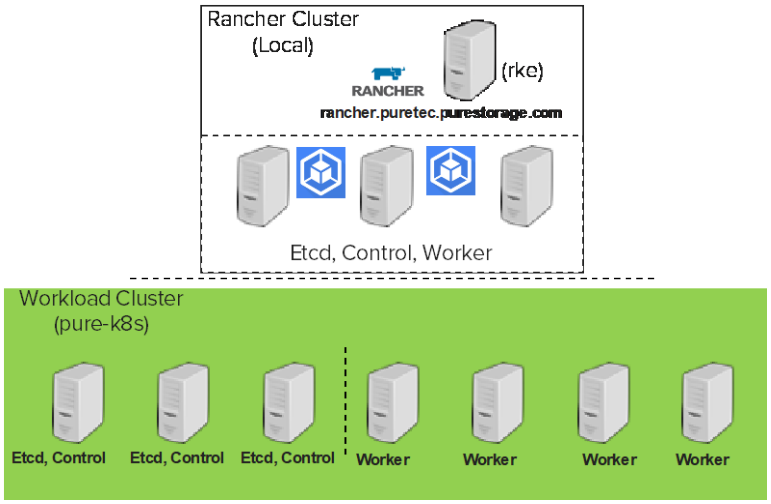

Figure 10. Seven-node Workload Cluster—"pure-new-solutions" added to Rancher Management Cluster (local)

Once the three Rancher nodes are up and running, you can add new workload clusters from "Add cluster" in the "Global" Rancher menu.

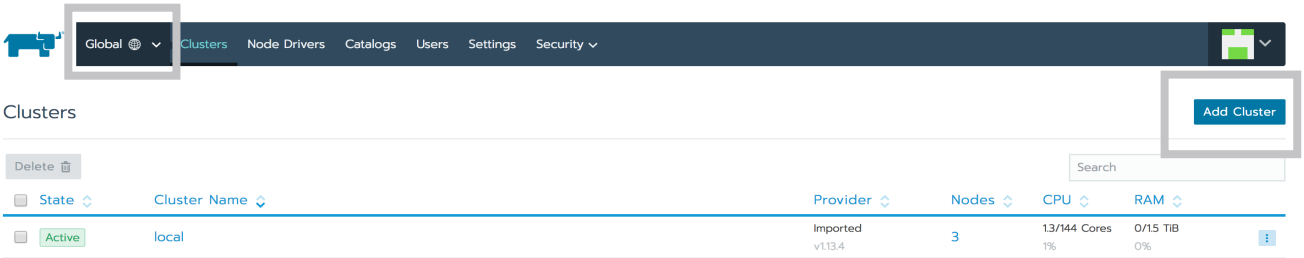

FIGURE 11. Add a new Workload Cluster from the Rancher global menu

Navigate from Rancher Global menu to "Add Cluster" for adding a new Workload cluster "pure-new-solutions."

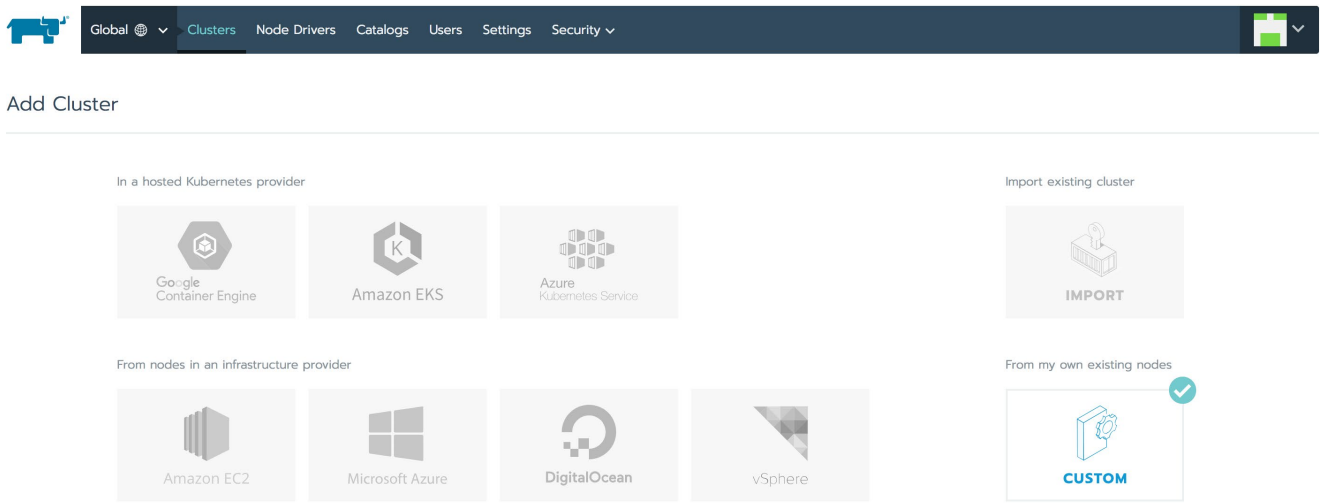

FIGURE 12. Select "Custom" to include all seven nodes for "pure-new-solutions" cluster

Provide a cluster name (pure-new-solutions in this example) and include the IP address of the node(s) that need to be added to this cluster and its role—etcd, control plane, or worker.

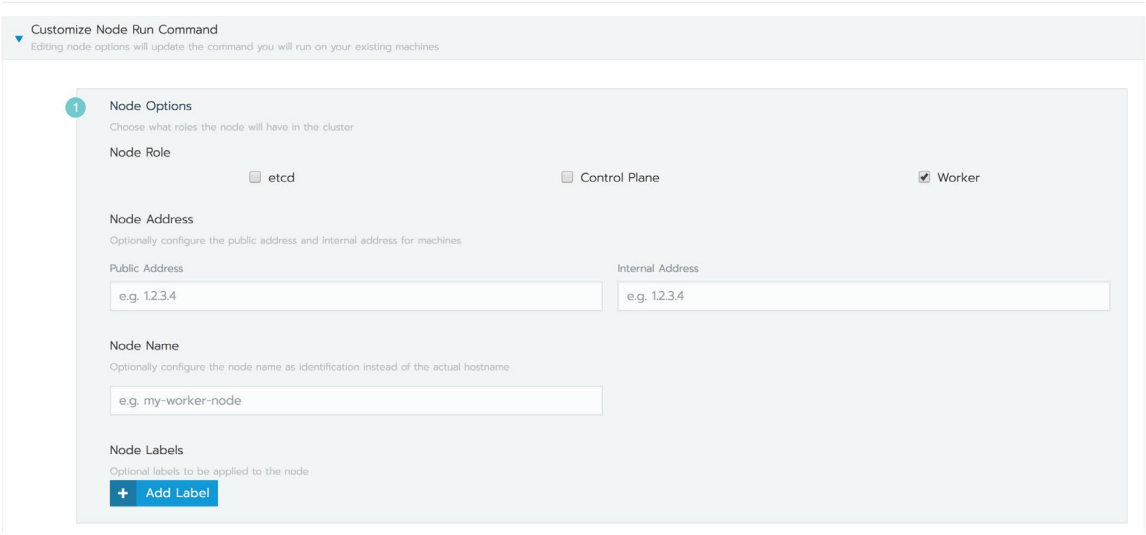

Figure 13. Entering all relevant data for each node

Rancher lists the command string at the bottom screen to run on the host that needs to be added to the pure-new-solutions cluster as etcd, control plane, and worker. The rancher agent is installed on the node and added to the pure-new-solutions cluster with its assigned role. Copy/paste the command from the Rancher window onto the node that has been added to the pure-new-solutions cluster.

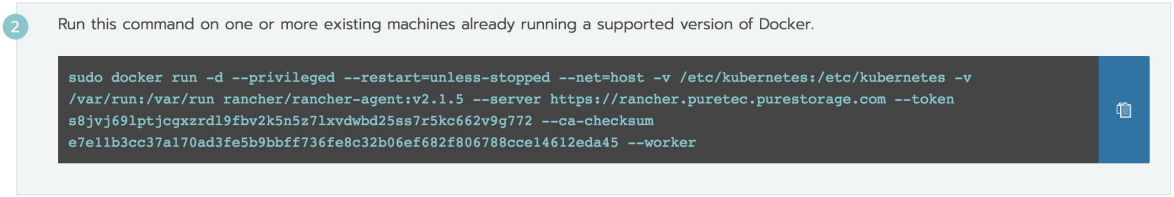

Figure 14. Adding a worker node to the pure-new-solutions cluster

```
[root@sn1-r720-g09-01 ~]# docker run -d --privileged --restart=unless-stopped --net=host -v
/etc/kubernetes:/etc/kubernetes -v /var/run:/var/run rancher/rancher-agent:v2.1.5 --server
https://rancher.puretec.purestorage.com --token s8jvj69lptjcgxzrdl9fbv2k5n5z7lxvdwbd25ss7r5kc662v9g772 --ca-
checksum
7e11b3cc37a170ad3fe5b9bbff736fe8c32b06ef682f806788cce14612eda45 --node-name sn1-r720-g09-01 --address
10.21.236.107 --internal-address 192.16.0.107 -worker
Unable to find image 'rancher/rancher-agent:v2.1.5' locally
v2.1.5: Pulling from rancher/rancher-agent
84ed7d2f608f: Pull complete
be2bf1c4a48d: Pull complete
a5bdc6303093: Pull complete
e9055237d68d: Pull complete
ecda3d2d744e: Pull complete
c2a506194467: Pull complete
224a48e7f77b: Pull complete
7a6deb26f059: Pull complete
03ee61b2892f: Pull complete
Digest: sha256:ab1f06bcdd6d41f201cfd423d44c0047525d7547e76635e57afb096322392757
Status: Downloaded newer image for rancher/rancher-agent:v2.1.5
e770831b46a7624bb453365c47caf30fc3e1ce01455e964435cdf750ec4cdf3f
```

A similar process is performed with the remaining six nodes to include them to the workload cluster pure-new-solutions. The new “pure-new-solutions” cluster will be listed in the Rancher global menu.

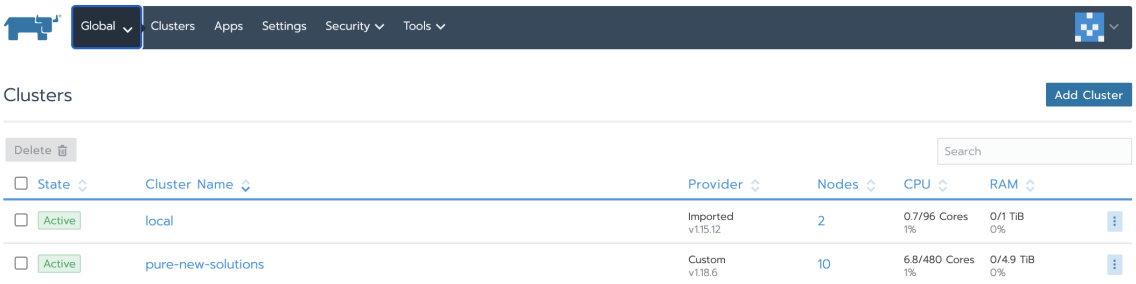

Figure 15. A new workload cluster “pure-new-solutions” is added to the Rancher Management Cluster

All seven nodes are listed as part of the pure-new-solutions cluster.

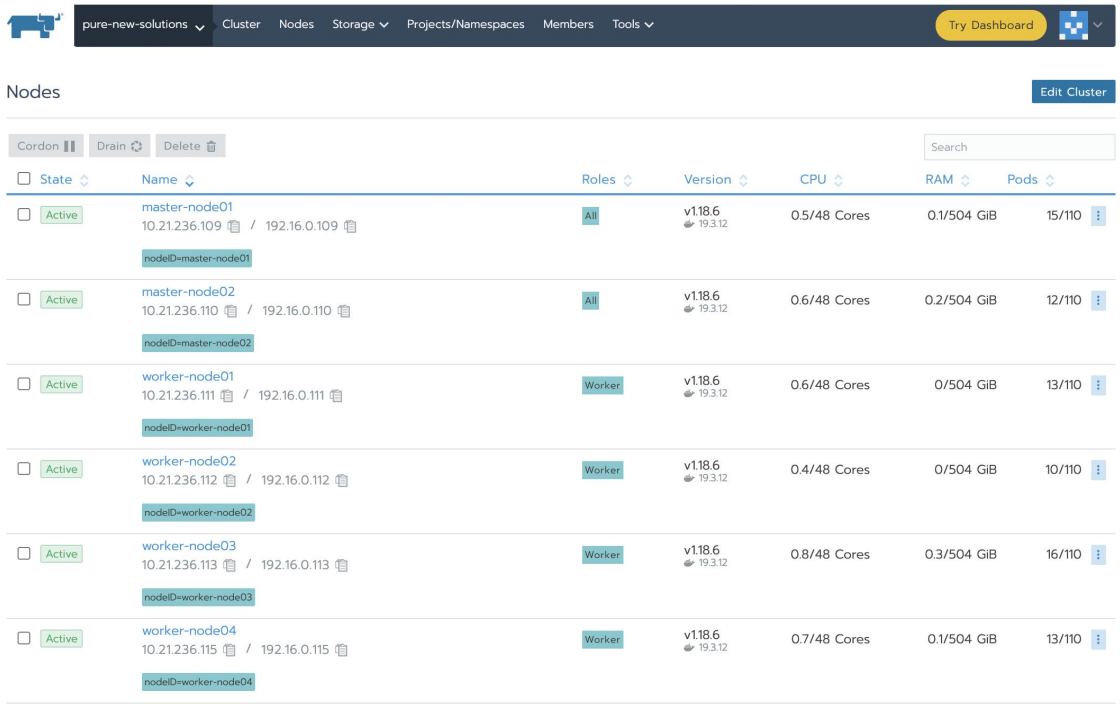

Figure 16. All seven nodes added to the pure-new-solutions cluster

Dynamic Provisioning of Persistent Volumes for Storage Disaggregation

Stateful applications like databases, source code, and build repositories—or business and technical applications that generate structured or unstructured data—all require persistent storage in the development and deployment process. As you provide more autonomy to end-users, consuming persistent storage for applications should be automated, dynamic, and disaggregated from compute. That means administrators can scale applications and storage independently and on-demand.

Pure Service Orchestrator

Pure Storage is a leader in flash technology and provides products with low latency, high IOPS, and high bandwidth, like Pure FlashArray and FlashBlade products. FlashArray supports iSCSI and FC protocols with very low latency and robust data-management capabilities. FlashBlade provides a unique platform that supports both NFSv3/v4.1 and S3 object store for high IOPS and bandwidth workloads that require seamless performance and capacity scaling independent of application scaling.

While an increasing number of applications are deployed in Kubernetes for scalability and resiliency, Pure Service Orchestrator is designed to disaggregate the underlying storage—FlashArray and FlashBlade—for different latency- and bandwidth-sensitive workloads. Pure Service Orchestrator is an out-of-tree dynamic provisioner for different storage-class objects like block and file that supports Container Storage Interface (CSI) standards. Pure Service Orchestrator CSI driver exposes block and file storage class from FlashArray and FlashBlade respectively to stateful applications running in a Rancher Kubernetes cluster.

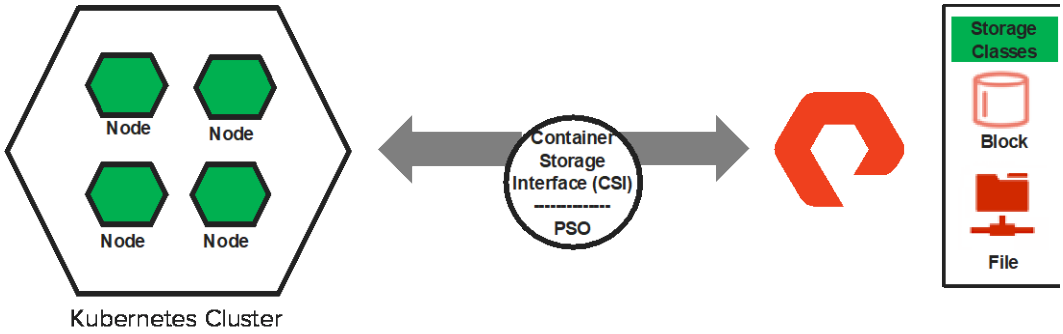

Figure 17. Out-of-tree dynamic provisioner

Therefore, Kubernetes ships an out-of-tree CSI driver that enables issue isolation from the Kubernetes cluster. It also allows data storage vendors like Pure Storage to release updated versions of Pure Service Orchestrator independent of Kubernetes releases.

Figure 17 shows an out-of-tree dynamic storage provisioner. Pure Service Orchestrator is currently available in the form of a helm chart that you can download and use in conjunction with a Kubernetes cluster in production.

The Container Storage Interface (CSI) is a standard that is formalized by CNCF and Pure Storage adopts the CSI features since Pure Service Orchestrator 5.x. The latest Pure Service Orchestrator v6.0.x is stateful that stores the metadata like volumes, hosts, IQNs of the physical host and the attachments, etc. for applications running in the Kubernetes pods. The Pure Service Orchestrator DB also contains the FlashArray and FlashBlade management endpoints and purity versions etc. information. Pure Service Orchestrator v6.0.x supports QoS for volumes created on FlashArray along with Snapshot and cloning capabilities.

Pure Service Orchestrator v6.0.x also allows you to configure export policies for filesystems along with custom storage classes that support different mount options for the filesystems provisioned on FlashBlade. Pure Service Orchestrator v6.0.x mounts the filesystems from FlashBlade over NFSv4.1 by default. A custom storage class with the required mount options can be created to mount the filesystem over NFSv3.

Key Pure Service Orchestrator Features

Pure Service Orchestrator is a dynamic provisioner that provisions persistent storage to stateful applications running in a Kubernetes cluster. Apart from provisioning persistent data storage to stateful applications on demand, Pure Service Orchestrator has the following key features:

- Full integration with Kubernetes and Docker
- Seamless scaling across multiple storage arrays
- Policy-based provisioning
- Data resiliency and self-healing capabilities for transparent recovery

Figure 18 illustrates the Rancher and Pure Service Orchestrator architecture. The dynamic provisioner runs as a pod in the pure-new-solutions cluster and is responsible for provisioning the Storage class—block (FlashArray) and file (FlashBlade). The Pure CSI-driver runs on each worker node that is part of the working cluster. The CSI driver is responsible for create/delete, mount/unmount, and attach/detach of persistent volumes for containers/pods.

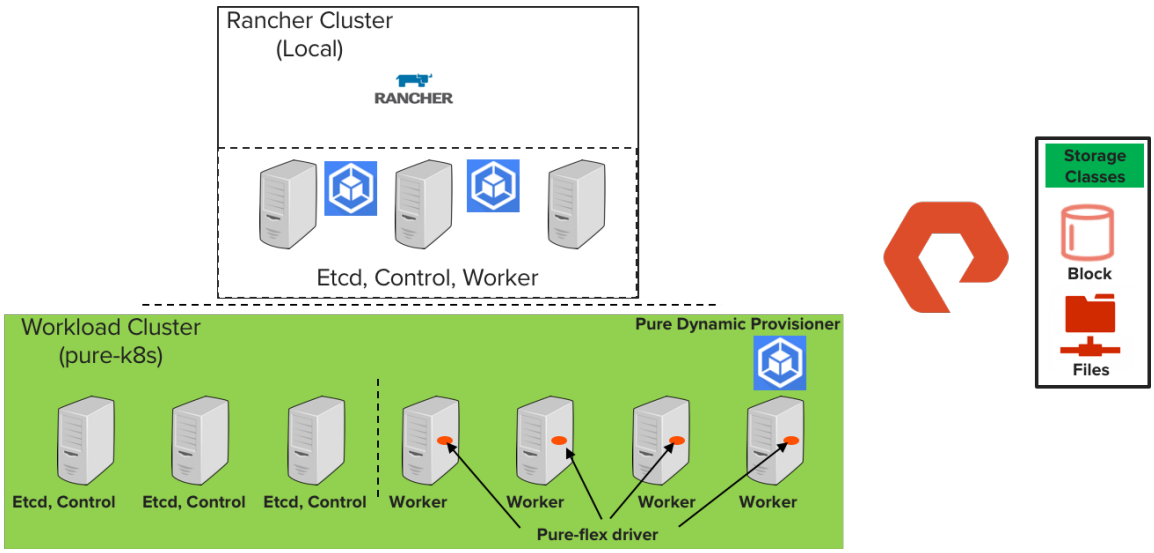

Figure 18. Pure Service Orchestrator layout in purek8s cluster

Using Helm charts is the preferred and easiest way to install Pure Service Orchestrator. Installing Pure Service Orchestrator using Rancher is further simplified. Pure Service Orchestrator can be configured one time using the menu as a part of the Rancher catalog. Under “Catalog” in the “Global” menu, select “Add Catalog.”

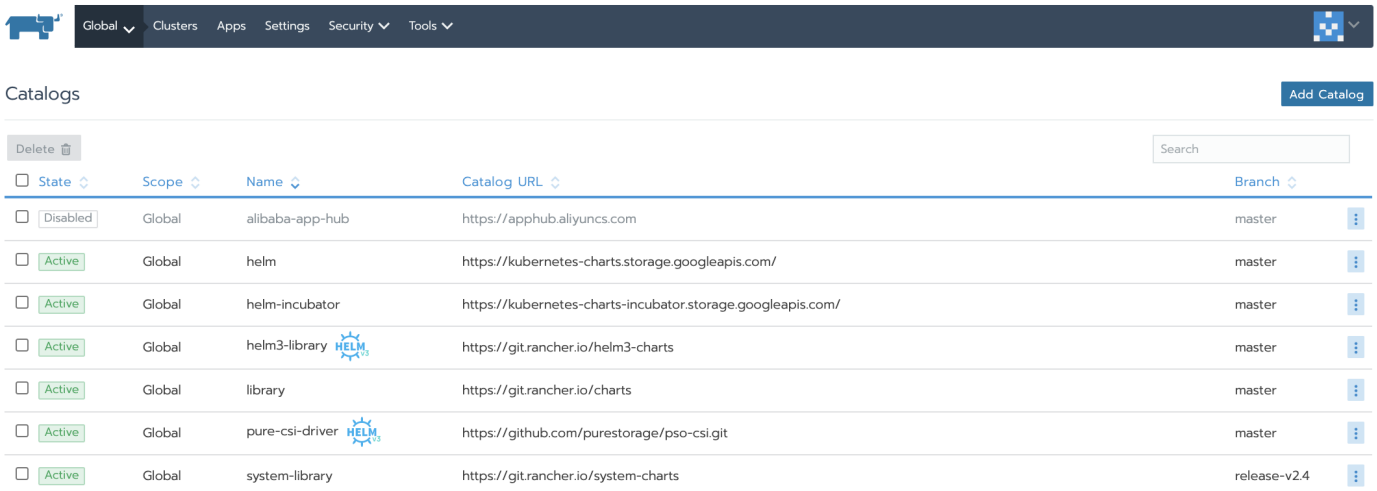

Figure 19. Catalog and stable application from Rancher global menu

Selecting “Add Catalog” will bring up a sub-menu that will require a name and the Pure Service Orchestrator’s Helm chart location. Select “Helm v3” for installing the Pure Service Orchestrator 6.0.x chart.

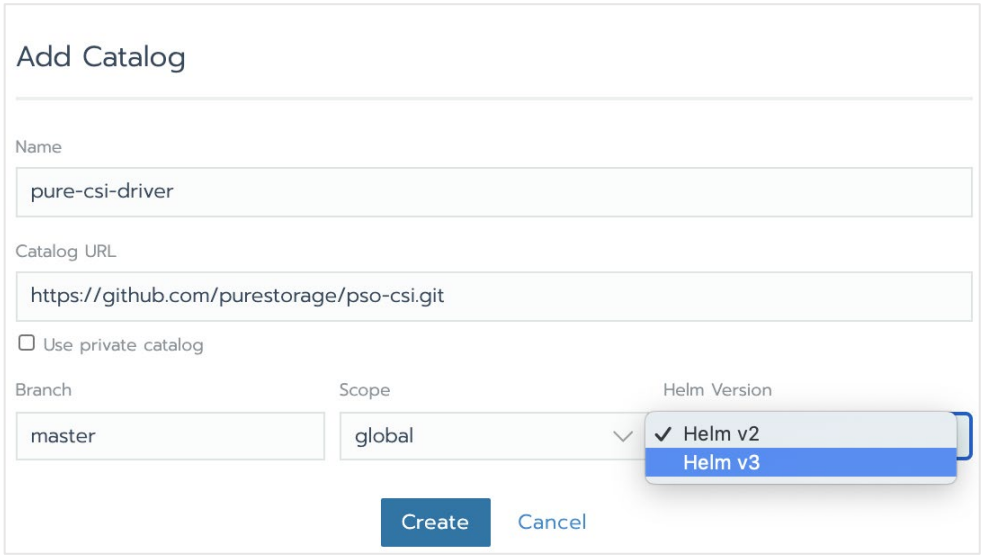

Figure 20. Adding the Pure Service Orchestrator Helm chart to the Rancher custom catalog

Once the Pure Service Orchestrator is added to the catalog, install Pure Service Orchestrator in the new workload cluster created. It is recommended to use the default namespace “pure-pso” to install the PSO-CSI v6.0.x driver. The pure-csi-driver will appear under “Catalog Apps” in the Rancher menu.

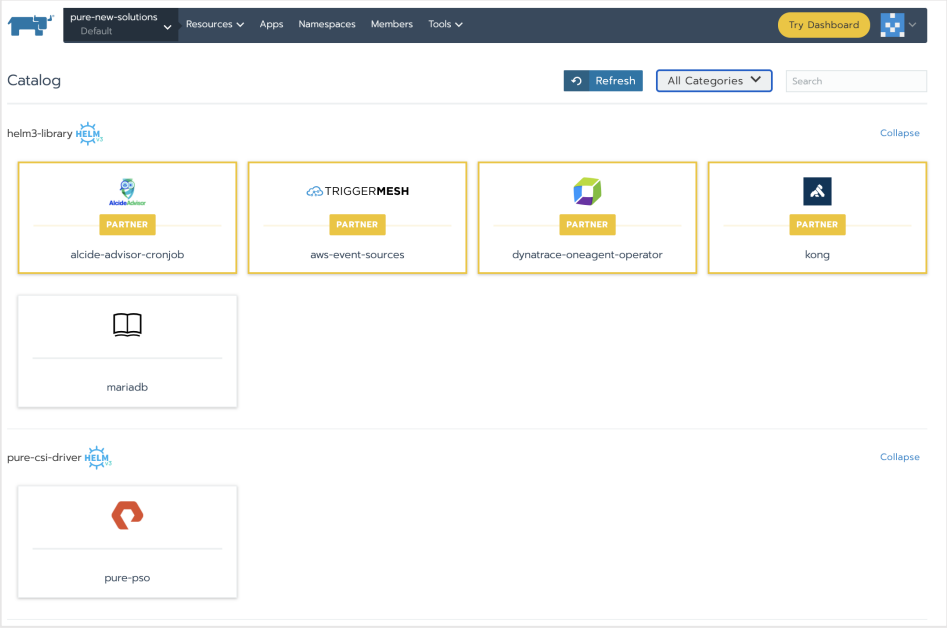

Figure 21. PSO listed as a catalog application

At the time of this writing, v6.0.1 was the latest Pure Service Orchestrator version available. Rancher allows you to upgrade to the latest version from its drop-down menu. The “View Details” link under “pure-csi-driver” leads to the next page, where you can paste the values.yaml file as “Edit as YAML.”

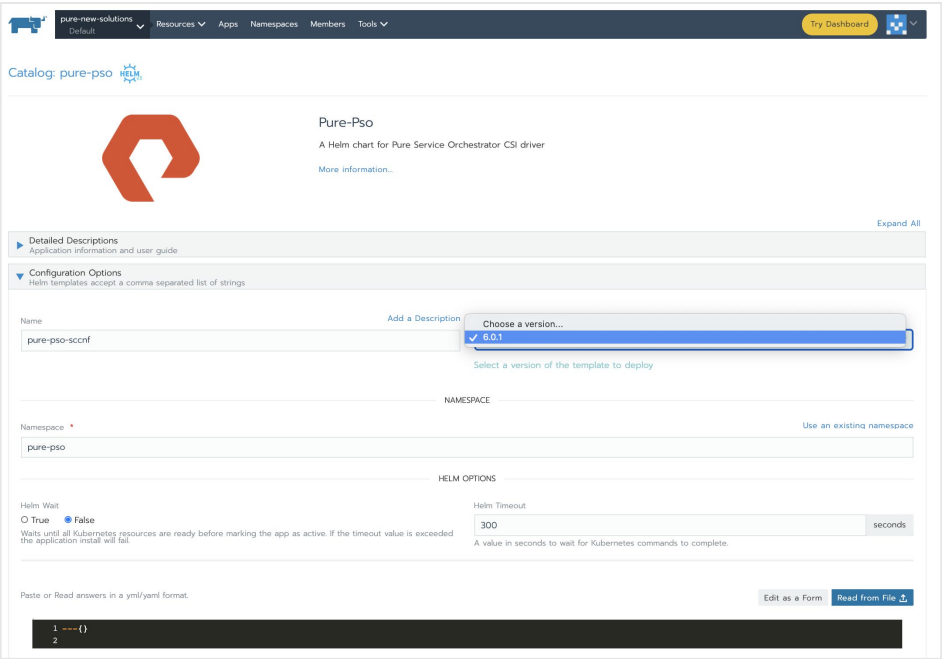

Figure 22. One-time Pure Service Orchestrator setup from Rancher

Before installing Pure Service Orchestrator, the following configurations are required on the Rancher management cluster (local) nodes and on the FlashArray:

1. Install “iscsi-initiator-utils” and “multipath” tools on all the Rancher management Cluster (local) and Workloads cluster (pure-new-solutions) nodes as a pre-requisite. In this example, there are three Rancher and seven pure-new-solutions cluster nodes.

```
[root@sn1-r720-g09-09 ~]# yum install iscsi-initiator-utils

[root@sn1-r720-g09-09 ~]# cat /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.1994-05.com.redhat:327c4ddd9fe
[root@sn1-r720-g09-09 ~]# chkconfig iscsid on
Note: Forwarding request to 'systemctl enable iscsid.service'.
Created symlink from /etc/systemd/system/multi-user.target.wants/iscsid.service to
/usr/lib/systemd/system/iscsid.service.
[root@sn1-r720-g09-09 ~]# systemctl enable iscsid.service
[root@sn1-r720-g09-09 ~]# systemctl start iscsid.service
[root@sn1-r720-g09-09 ~]# systemctl status iscsid

yum install device-mapper-multipath
mpathconf --enable --user_friendly_names y
```

2. For iSCSI volumes with a Rancher-launched Kubernetes cluster (pure-new-solutions) also known as workload cluster, you need to add additional lines to the cluster yaml file. Edit the cluster “pure-new-solutions” to include the following lines under “services:”

```
services:
  kubelet:
    extra_binds:
      - "/etc/iscsi:/etc/iscsi"
      - "/sbin/iscsiadm:/sbin/iscsiadm"
```

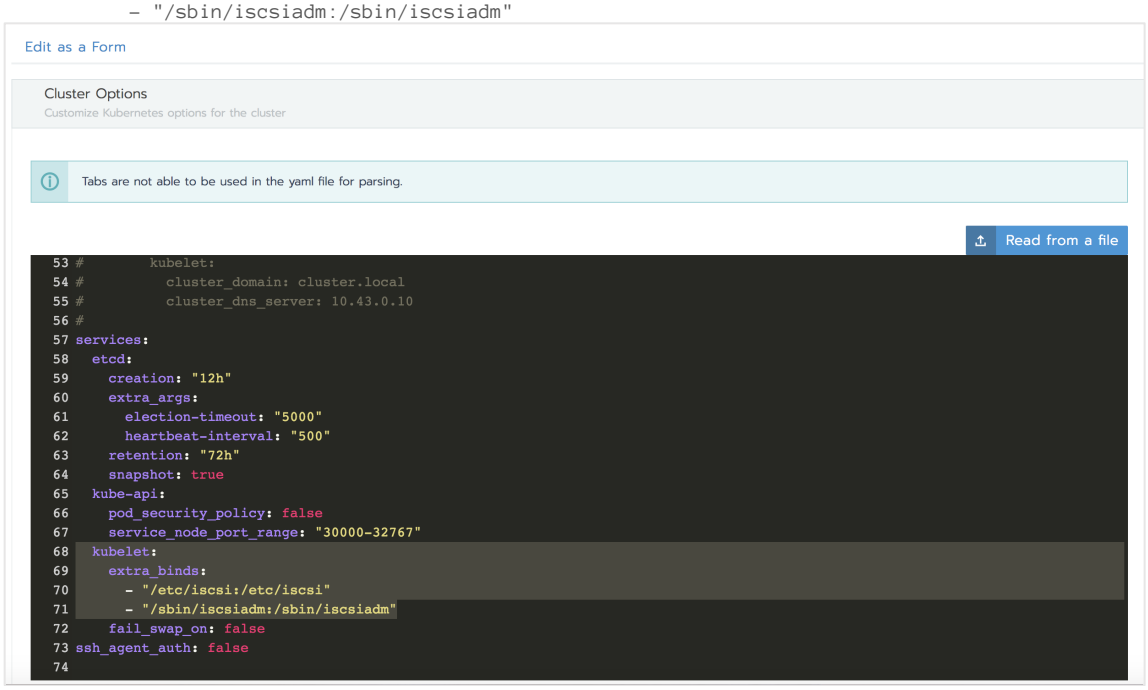

Figure 23. Additional settings for iSCSI volumes on FlashArray

Note: No additional configuration or changes are required for FlashBlade and Rancher except for the “nfs-utils” RPM installed on the Centos 7 nodes to enable NFS functionality.

You can customize the following values.yaml file with respect to FlashArray, FlashBlade, and other environment variables like running a database on some select cluster nodes etc.

```
# Default values for pureStorageDriver.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

# clusterID is added as a prefix for all volumes created by this PSO installation.
# clusterID is also used to identify the volumes used by the datastore, pso-db.
# clusterID MUST BE UNIQUE for multiple k8s clusters running on top of the same storage arrays.
# NOTE: in previous versions, this property was called "namespace.pure"
# RESTRICTIONS:
#  max length: 22
#  characters allowed: alphanumeric and underscores
clusterID: purenewsolutions

# support k8s or openshift
orchestrator:
  # name is either 'k8s' or 'openshift'
  name: k8s

# this option is to enable/disable the debug mode of this app
# for pso-csi
app:
  debug: false

# this option is to enable/disable the csi topology feature
# for pso-csi
storagetopology:
  enable: false
  strictTopology: false

# arrays specify what storage arrays should be managed by the plugin, this is
# required to be set upon installation. For FlashArrays you must set the "MgmtEndPoint"
# and "APIToken", and for FlashBlades you need the additional "NFSEndPoint" parameter.
# The labels are optional, and can be any key-value pair for use with the "fleet"
# provisioner. An example is shown below:
arrays:
  FlashArrays:
    - MgmtEndPoint: "10.21.126.20"
      APIToken: "d6477e15-dc61-0d43-e863-cd66e2a936ea"
#       Labels:
#     - MgmtEndPoint: "1.2.3.5"
#       APIToken: "b526a4c6-18b0-a8c9-1afa-3499293574bb"
  FlashBlades:
    - MgmtEndPoint: "10.21.241.11"
      APIToken: "T-154d4220-3015-489c-8d64-fe7ea4f93499"
      NFSEndPoint: "10.21.236.100"
#       Labels:
#     - MgmtEndPoint: "1.2.3.8"
#       APIToken: "T-d4925090-c9bf-4033-8537-d24ee5669135"
#       NFSEndPoint: "1.2.3.9"
#arrays:
#  FlashArrays: []
#  FlashBlades: []

flasharray:
  # support ISCSI or FC, not case sensitive
  sanType: ISCSI
  defaultFSType: xfs
  defaultFSOpt: "-q"
  defaultMountOpt:
    - discard
```

```
    preemptAttachments: "true"
    iSCSILoginTimeout: 20

  flashblade:
    snapshotDirectoryEnabled: "false"
    exportRules: "*(rw,no_root_squash)"
  images:
    plugin:
      name: purestorage/k8s
      tag: v6.0.1
      pullPolicy: Always
    csi:
      provisioner:
        name: quay.io/k8scsi/csi-provisioner
        pullPolicy: Always
      snapshotter:
        name: quay.io/k8scsi/csi-snapshotter
        pullPolicy: Always
      attacher:
        name: quay.io/k8scsi/csi-attacher
        pullPolicy: Always
      resizer:
        name: quay.io/k8scsi/csi-resizer
        pullPolicy: Always
      nodeDriverRegistrar:
        name: quay.io/k8scsi/csi-node-driver-registrar
        pullPolicy: Always
      livenessProbe:
        name: quay.io/k8scsi/livenessprobe
        pullPolicy: Always
    database:
      deployer:
        name: purestorage/dbdeployer
        tag: v1.0.1
        pullPolicy: Always
      cockroachOperator:
        name: purestorage/cockroach-operator
        tag: v1.0.1
        pullPolicy: Always

  # specify the service account name for this app
  clusterrolebinding:
    serviceAccount:
      name: pure

  # Top-level nodeSelector, tolerations, and affinity will be applied to all PSO pods
  # Use the nodeSelector, tolerations, and affinity variables in the specific sections to override the top-
  level values
  # for each respective component

  # nodeSelector is the simplest way to limit which kubernetes nodes can be used for the pod
  nodeSelector: {}
  # disktype: ssd

  # tolerations allow pods to run on tainted kubernetes nodes
  tolerations: []
  # - operator: Exists

  # affinity provides more granular control of which kubernetes nodes will run the pod
  affinity: {}
  # nodeAffinity:
  #   requiredDuringSchedulingIgnoredDuringExecution:
  #     nodeSelectorTerms:
  #     - matchExpressions:
```

```
#       - key: e2e-az-NorthSouth
#         operator: In
#         values:
#         - e2e-az-North
#         - e2e-az-South

nodeServer:
  # nodeSelector is the simplest way to limit which kubernetes nodes will run the CSI node server
  # Please refer to the top-level description of nodeSelector for an example
  nodeSelector: {}
  # tolerations allow CSI node servers to run on tainted kubernetes nodes
  # Please refer to the top-level description of tolerations for an example
  tolerations: []
  # affinity provides more granular control of which kubernetes nodes will run the CSI node servers
  # Please refer to the top-level description of affinity for an example
  affinity: {}

controllerServer:
  # nodeSelector is the simplest way to limit which kubernetes node will run the CSI controller server
  # Please refer to the top-level description of nodeSelector for an example
  nodeSelector: {}
  # tolerations allows the CSI controller servers to run on a tainted kubernetes node
  # Please refer to the top-level description of tolerations for an example
  tolerations: []
  # affinity provides more granular control of which kubernetes node will run the CSI controller server
  # Please refer to the top-level description of affinity for an example
  affinity: {}

database:
  # nodeSelector is the simplest way to limit which kubernetes nodes will run the database-related pods
  # Please refer to the top-level description of nodeSelector for an example
  nodeSelector: {}
  # tolerations allows the database-related pods to run on tainted kubernetes nodes
  # Please refer to the top-level description of tolerations for an example
  tolerations: []
  # affinity provides more granular control of which kubernetes nodes will run the database-related pods
  # Please refer to the top-level description of affinity for an example
  affinity: {}

  # resources is used specifically for the database pods
  # Change these to appropriate values for the hardware that you're running. You can see
  # the amount of allocatable resources on each of your Kubernetes nodes by running:
  #   kubectl describe nodes
  resources: {}
  # requests:
  #   cpu: "16"
  #   memory: "8Gi"

  # The length of time a db node is considered Suspect before being marked as Down
  # This value is recommended to be larger than Kubernetes liveness probe setting time in our db pod,
  # so K8S gets chance to restart the pod if it's unhealthy before we take it down.
  # https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-
probes/#configure-probes
  maxSuspectSeconds: 600
  # The length of time a db node is allowed to be in Startup before being suspect
  maxStartupSeconds: 300
```

Get the api-token for FlashArray and FlashBlade by logging into the Pure Storage appliances using SSH and CLI or from the FlashArray GUI. If the api-token is not available, a new one should be created. Add the appropriate api-token as shown in the above values.yaml file.

FlashArray:

```
$ ssh pureuser@10.21.126.20
pureuser@10.21.126.20's password:
X11 forwarding request failed on channel 0
Last login: Tue Jan 15 16:35:36 2019 from 192.168.3.4

Fri Feb 22 11:30:52 2019
Welcome pureuser. This is Purity Version 5.1.2 on FlashArray sn1-x70-d06-25
http://www.purestorage.com/
pureuser@sn1-x70-d06-25> pureadmin list --api-token pureuser --expose
Name      Type   API Token                               Created                  Expires
pureuser  local  d6477e15-dc61-0d43-e863-cd66e2a936ea  2019-01-11 15:49:44 PST  2020-01-11 15:49:44 PST
pureuser@sn1-x70-d06-25>
```

FlashBlade:

```
$ ssh pureuser@10.21.241.11
pureuser@10.21.241.11's password:
X11 forwarding request failed on channel 0
Welcome to Iridium
(sn1-fb-g06-01-ch1-fm1  3.16.0-44-iridium  #59~14.04.1 SMP PREEMPT Fri Nov 16 23:32:05 UTC 2018)
Last login: Wed Jan 16 00:22:14 2019 from 192.168.3.4
pureuser@sn1-fb-g06-01-ch1-fm1> pureadmin list --api-token pureuser --expose
Name      API Token                               Created                  Expires
pureuser  T-1cf8cc93-0e34-4cce-9d91-d03b67c2c890  2018-10-29 01:30:16 EDT  2019-10-29 01:30:16 EDT
pureuser@sn1-fb-g06-01-ch1-fm1>
```

After launching Pure Service Orchestrator from the Rancher menu, the CSI driver should be running on all the worker nodes and the dynamic provisioner should be running as a pod in the pure-new-solutions cluster. The stateful Pure Service Orchestrator v6.0.x spins up five database pods by the pso-db-deployer. The five database pods create persistent volumes on FlashArray or FlashBlade endpoints. The pso-db-cockroach-operator pod checks the health of the DB pods and keeps them in sync. The pso-csi-node daemonset runs on all the worker nodes in the Rancher Kubernetes cluster. This is responsible to mount/unmount, attach/detach and create/delete volumes and filesystems from FlashArray and FlashBlade respectively. The pso-csi-controller interfaces with the Kuberentes CSI driver.

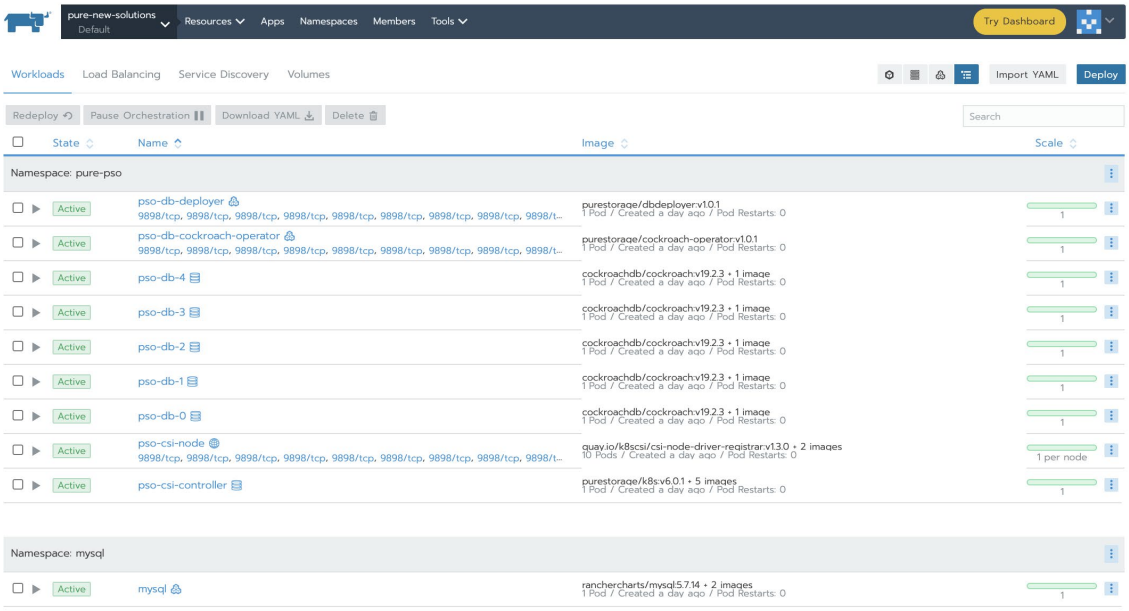

Figure 24. Pure Service Orchestrator CSI driver running as "pod" in the pure-kk8s cluster

Now the Pure Storage classes are listed, and the default storage class is set from the Rancher menu to be used by any application launched in the working cluster as shown in the figure below.

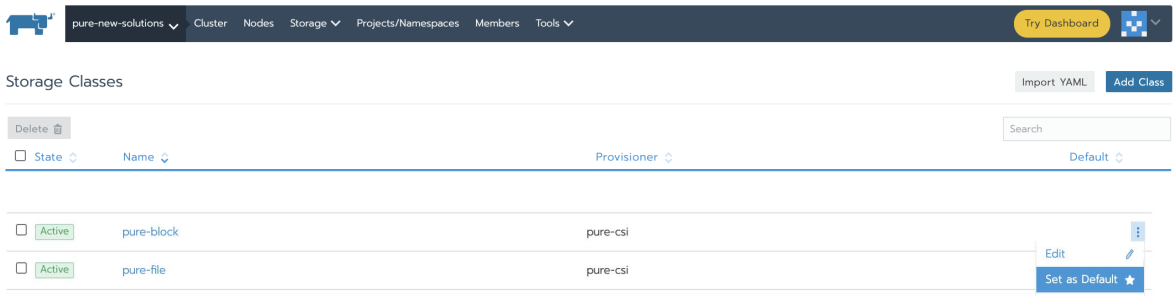

Figure 25. Pure Storage classes listed in Rancher

The storage classes available to the pure-new-solutions cluster are now listed in Rancher: pure-block is for PVs over FC and iSCSI; pure-file is for PVs requested over NFS. The Kubernetes cluster was operational in less than five minutes, as mentioned earlier in this paper. Pure Service Orchestrator is now ready to dynamically manage and provision PVs on-demand by users and by the application.

Before we start using Pure Service Orchestrator, there are two main components to how a user or an application would request or access persistent storage from the Kubernetes cluster: persistent volume claim and persistent volume.

Persistent volume claim: PVC is a request to the dynamic provisioner for persistent storage by a user or an application. PVC requests for a provisioner include size, access mode (RWO, RWM), and storage class for the persistent storage or volume.

Persistent volume: PV is the physical storage space accessed over the block or file that stores persistent data for an application. A pod consists of a container that attaches and mounts to a PV. The pure CSI driver is responsible for attaching and mounting a PV based on the PVC's storage class.

Pure Service Orchestrator Workflow in a Kubernetes Cluster

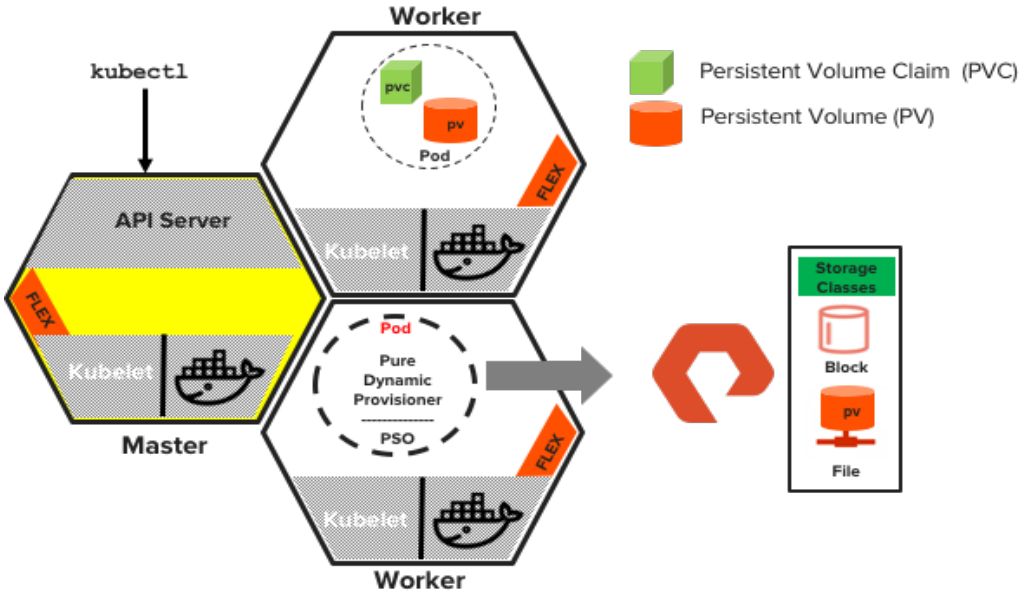

Figure 26. Pure dynamic provisioner and Kubernetes cluster

As depicted in Figure 26, the following steps comprise the communication between the PVC and the dynamic provisioner when the user or application requests a PV.

1. The user or the application creates a PVC using the *-pvc.yaml file.
2. The Kubernetes PVC provisioner gets the request to create the PV.
3. The PVC controller looks for a PV that it can bind to while it waits for a PV to be available.
4. In the meantime, the Pure dynamic provisioner is notified of the PVC that was created. It checks whether the storage class specified in the *-pvc.yaml file matches with its supported classes and provisions the PV as needed.
5. Creating the PV entails calling into Pure Service Orchestrator to create the actual volume or filesystem on FlashArray or FlashBlade respectively. Once the PV is created, the object details are pushed to the API Server.
6. The PVC controller now finds the newly created object and can bind the PV to the PVC.

The following steps illustrate the Pure CSI driver's role in Pure Service Orchestrator:

1. The Kubernetes CSI driver mounts the NFS share or the PV to the mount path mentioned in the *-pod.yaml.
2. The kubelet sees the pod needs a volume mounted, looks up the volume, and finds that the PVC is bound to a PV that specifies a CSI volume.
3. Kubelet looks up the CSI volume plugin associated with the PV.

4. Kubelet calls the Pure CSI driver to mount the PV.
5. The Pure CSI driver looks up the PV to find the array to attach. It finds FlashBlade.
6. The Pure CSI driver finds the attached volume (file-system) and mounts it over NFS.
7. Kubelet passes the mount path to the Container Runtime Interface (CRI) plugin—Docker in this case. The CRI for Docker handles mounting it once more into the container as defined in the container's yaml specifications.

The application image specified in the *-pod.yaml is installed.

A service is created with an external IP address and port number to provide public access.

Conclusion

The Kubernetes ecosystem continues to grow by leaps and bounds, providing more robustness, security, and automated service discovery. Simplifying some of the basic operations, like cluster setup and configuration—along with dynamically provisioning persistent storage using Rancher and Pure Service Orchestrator—enables admins to accelerate Kubernetes environment readiness for developers and end-users, enabling seamless consumption of infrastructure. Production-ready installers allow admins to set up a basic Kubernetes cluster with all its dependencies very quickly. Pure Service Orchestrator, with its simple setup processes and native availability within Rancher, enables end users to scale application and data independently in a disaggregated manner.

About the Authors

Bikash Roy Choudhury, Technical Director for DevOps/EDA, is responsible for designing and architecting solutions for DevOps workflows relevant across industry verticals including high tech, financial services, gaming, social media, and web-based organizations. He has also worked on validating solutions with Rancher/Kubernetes, GitLab, Jenkins, JFrog Artifactory, IBM Cloud Private and Perforce using RESTful APIs and integrating them with data platforms in private, hybrid, and public clouds. In his current role, Bikash drives integrations with strategic DevOps partners, including Rancher, Mesosphere, Perforce, GitLab, and JFrog.

Chris Kim is a Field Engineer at Rancher Labs based out of their Cupertino, CA Headquarters. He specializes in Rancher 2 and Kubernetes and has production experience running Kubernetes in the wild. Chris has experience architecting all types of Kubernetes clusters, ranging from purpose-built graphical compute clusters to general-purpose shared compute clusters. He also has experience as a developer, working in many languages from PHP to Go.

Pure Storage, Inc.
650 Castro Street, #400
Mountain View, CA 94041