PURESTORAGE® | RANCHER®

WHITE PAPER

RANCHER AND PURE SERVICE ORCHESTRATOR

SIMPLIFY IMPLEMENTATION FOR STATEFUL APPLICATIONS AND DISAGGREGATE STORAGE

TABLE OF CONTENTS

INTRODUCTION

As modern applications are increasingly modularized and consumed as microservices, containers are the common unit of deployment. But microservices running in naked containers are exposed to 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.

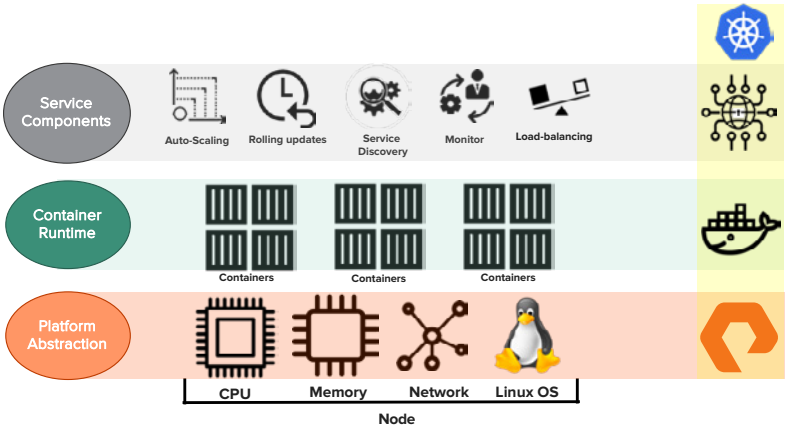

FIGURE 1. Container vs. Orchestrator

While Docker is one of the most frequently used container runtimes in production, Open Source Kubernetes is most common among the container orchestrators. Various different 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 in nature: Whenever a container is destroyed, the data associated with the application in that container is lost. 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™ (PSO) to provision persistent storage on Pure Storage solutions. The external provisioner runs in a pod in the Kubernetes cluster and uses a FLEX-based volume plugin to mount/unmount and attach/detach persistent volumes. The FLEX-based driver runs on all the 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. PSO 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 PSO from the Rancher catalog. PSO is an external persistent-storage provisioner from Pure Storage that runs in the Kubernetes cluster and that deploys stateful application workloads in a pod with persistent storage provisioned on Pure Storage FlashArray™, FlashBlade™, and Cloud Block Store products.

ANATOMY OF A KUBERNETES CLUSTER

As an orchestrator, Kubernetes has the ability to 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.

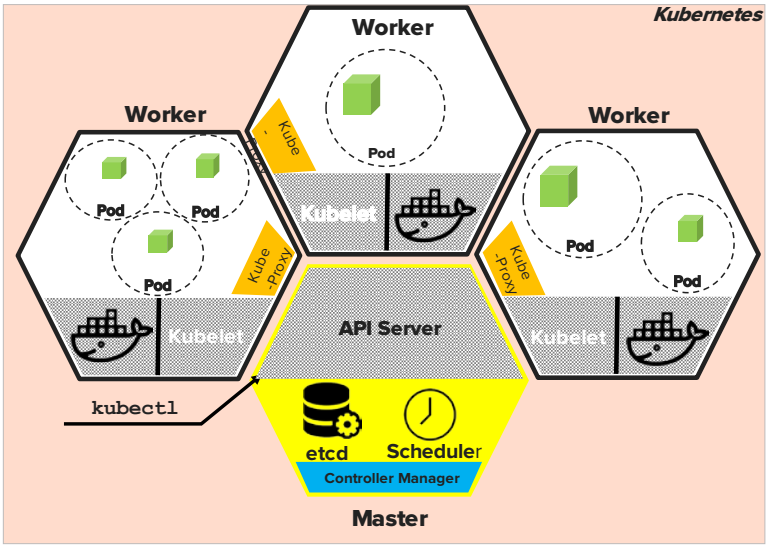

FIGURE 2. Four-node Kubernetes cluster

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 control plane and runtime components. The Control plane that runs in the master node primarily consists of the following components:

- **API server** – The management hub for the entire Kubernetes cluster. This server 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 all of 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).

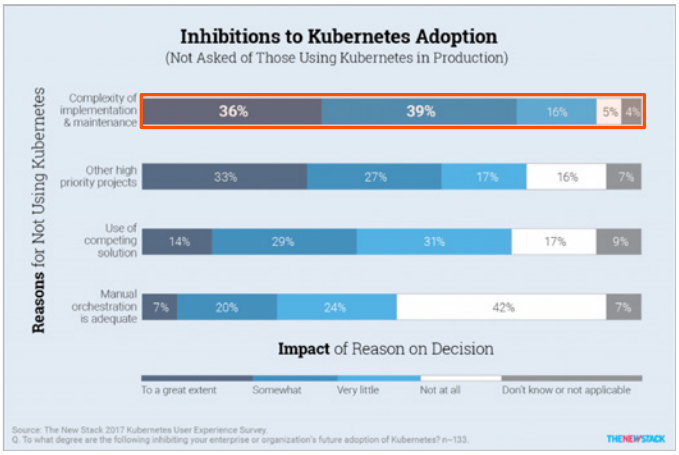

FIGURE 3. Top challenges with Kubernetes implementation

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 the purposes of 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 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 to the official Rancher documentation.

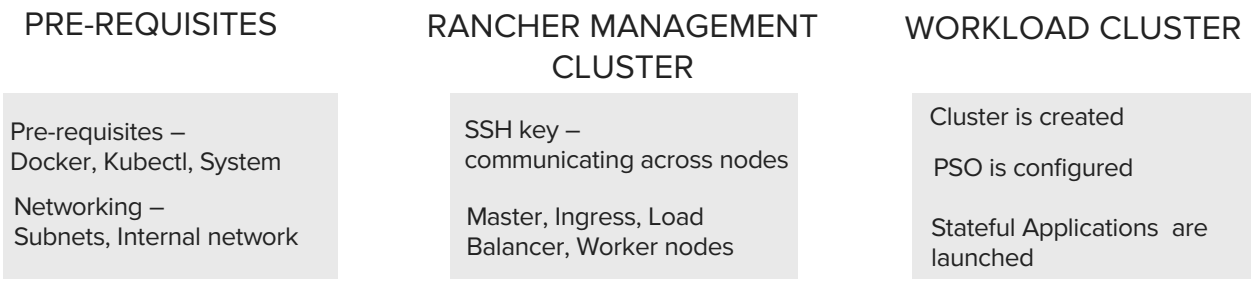

FIGURE 5. Kubernetes cluster creation.

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 (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.

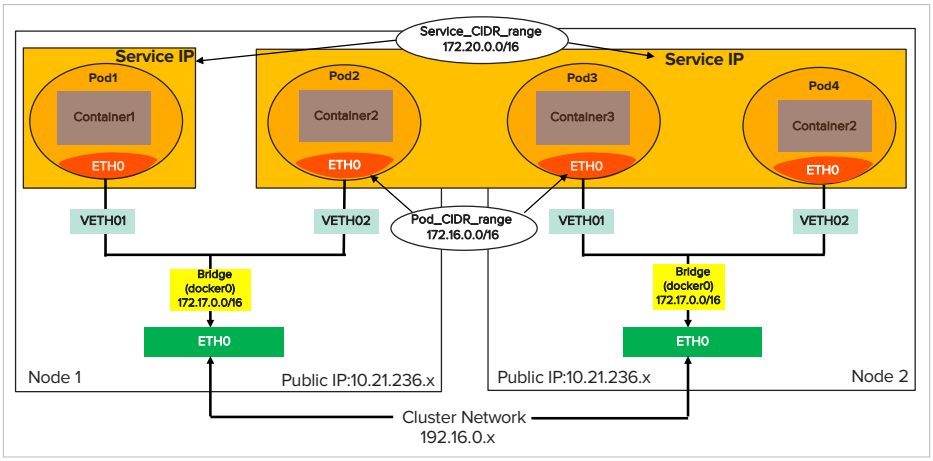

FIGURE 6. Basic network layout for services, pods, and nodes

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                    swap    defaults        0 0
```

Install Docker 18.09.2 on all the nodes that are part of the Rancher and workload clusters. This version of 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 core, 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.

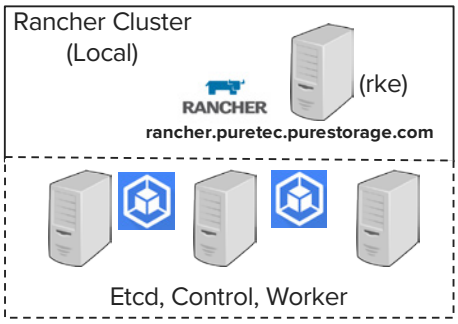

FIGURE 7. Three-node Rancher Management cluster with RKE running on a separate node

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 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
```

At this time, 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 and 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.purestorage.com to launch the cluster.

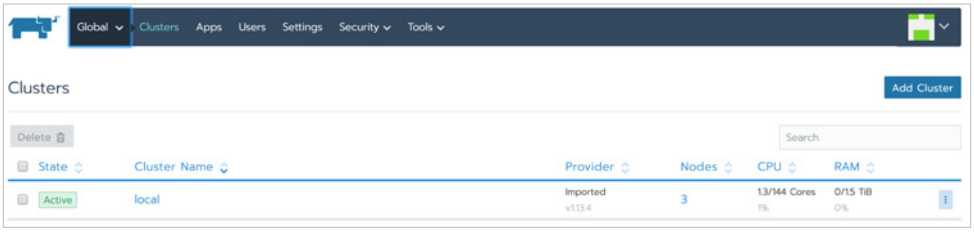

FIGURE 8. The "local" Rancher Management Cluster listed in the Rancher GUI

The three nodes are listed in the **local** cluster that represent the Rancher Management Cluster.

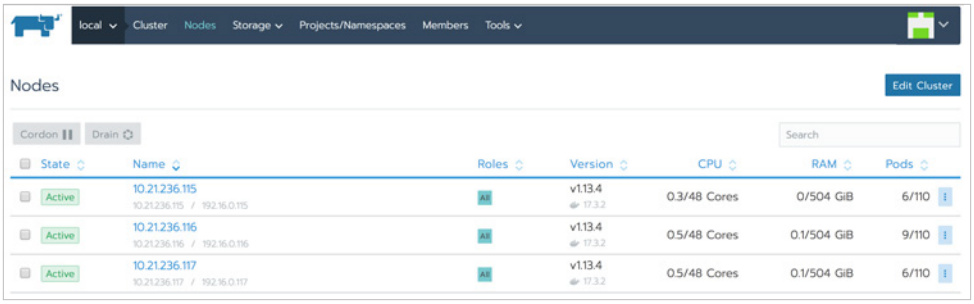

FIGURE 9. Three-node Rancher Management cluster (recommended)

Phase 3: Workload Cluster

At the time of writing, a brand new, seven-node workload cluster called **pure-k8s** is added to the Rancher management cluster (local). The workload cluster is responsible for running any container-based applications in the pure-k8s 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.

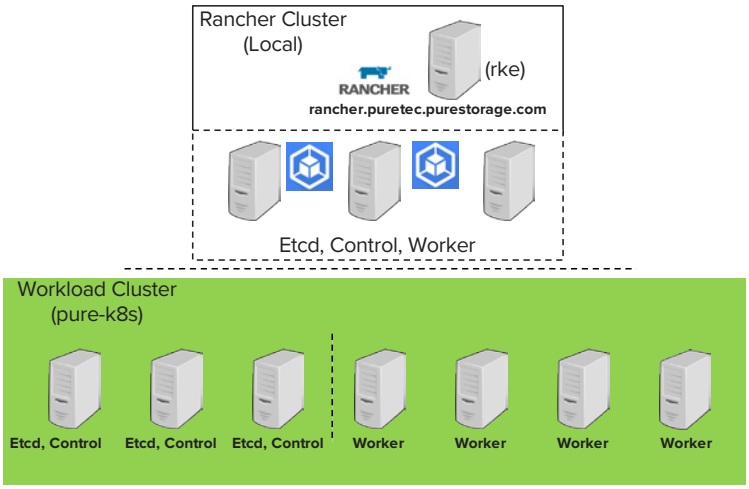

FIGURE 10. Seven-node Workload Cluster – "pure-k8s" added to Rancher Management Cluster (local)

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

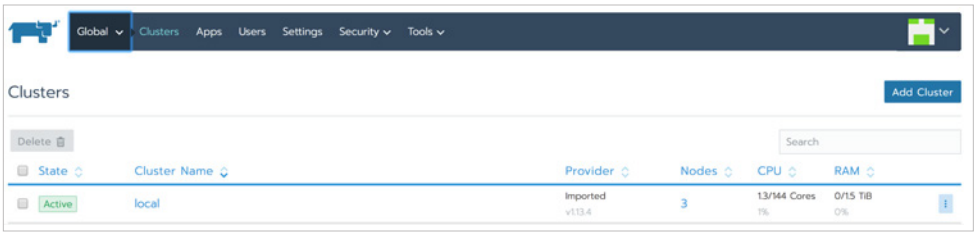

FIGURE 11. Add a new Workload Cluster from the Rancher Global Menu

Navigate from Rancher Global menu to **Add Cluster** for adding a new Workload cluster **pure-k8s**.

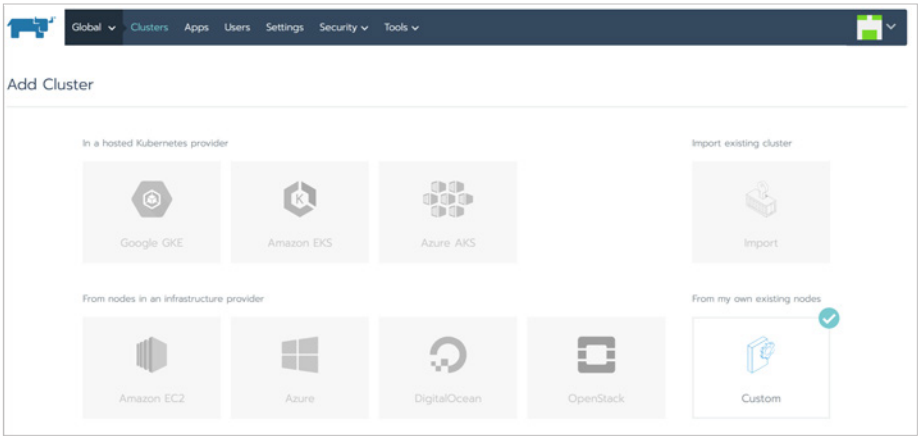

FIGURE 12. Select "Custom" to include all seven nodes for "pure-k8s" cluster

Provide a cluster name (pure-k8s 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**.

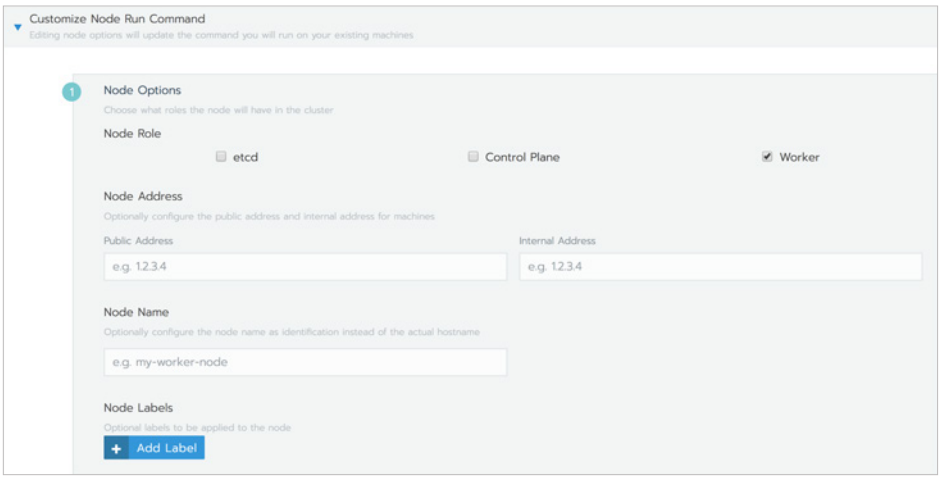

FIGURE 13. Entering all relevant data for each node

Rancher lists the command string at the bottom screen to run on the host that needs to be added to pure-k8s cluster as etcd, control plane, and worker. The rancher agent is installed on the node and added to the pure-k8s cluster with its assigned role. Copy/paste the command from the Rancher window onto the node that has been added to the pure-k8s cluster.

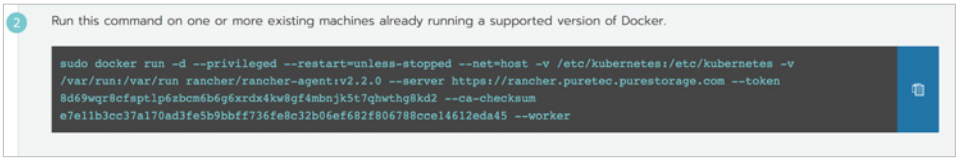

FIGURE 14. Adding a worker node to the pure-k8s cluster

```
[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-k8s. The new **pure-k8s** cluster will be listed in the Rancher global menu.

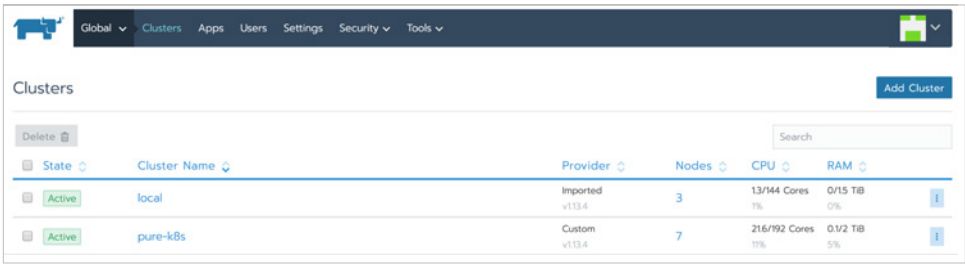

FIGURE 15. A new workload cluster "pure-k8s" is added to the Rancher Management Cluster

All seven nodes are listed as part of the **pure-k8s** cluster.

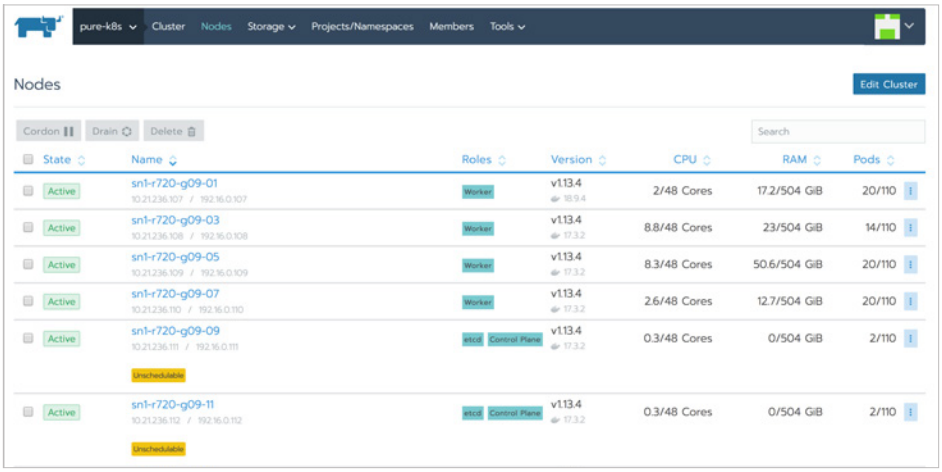

FIGURE 16. All seven nodes added to the pure-k8s cluster

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 (PSO)

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, PSO is designed to disaggregate the underlying storage – FlashArray and FlashBlade – for different latency- and bandwidth-sensitive workloads. PSO is an out-of-tree dynamic provisioner for different storage-class objects like block and file. (Storage-class objects have been supported since Kubernetes 1.4.)

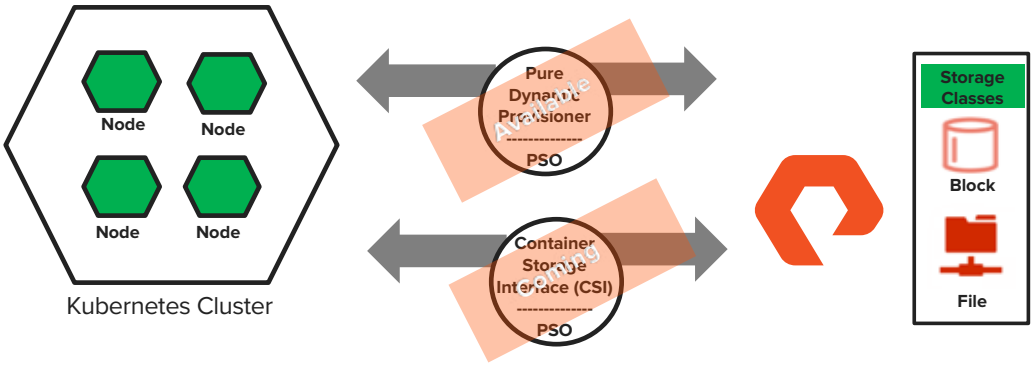

FIGURE 17. Out-of-tree dynamic provisioner

The in-tree FLEX volume driver plug-in is compiled and shipped with Kubernetes. As the plug-in is tightly coupled with Kubernetes, issues and bugs crash the entire Kubernetes cluster rather than limiting the impact to the plugin itself. Therefore, Kubernetes began to ship an out-of-tree FLEX volume driver plug-in that enables issue isolation from the Kubernetes cluster. It also allows data storage vendors like Pure Storage to release updated versions of PSO independent of Kubernetes releases.

Figure 17 shows an out-of-tree dynamic storage provisioner. PSO 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 new standard that is currently formalized by CNCF. Pure Storage plans to release a CSI later in2019.

Key PSO Features

PSO 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, PSO 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 PSO architecture. The dynamic provisioner runs as a pod in the pure-k8s cluster and is responsible for provisioning the Storage class – block (FlashArray) and file (FlashBlade). The Pure FLEX-driver runs on each worker node that is part of the pure-k8s cluster. The FLEX driver is responsible for create/delete, mount/unmount, and attach/detach of persistent volumes for containers/pods.

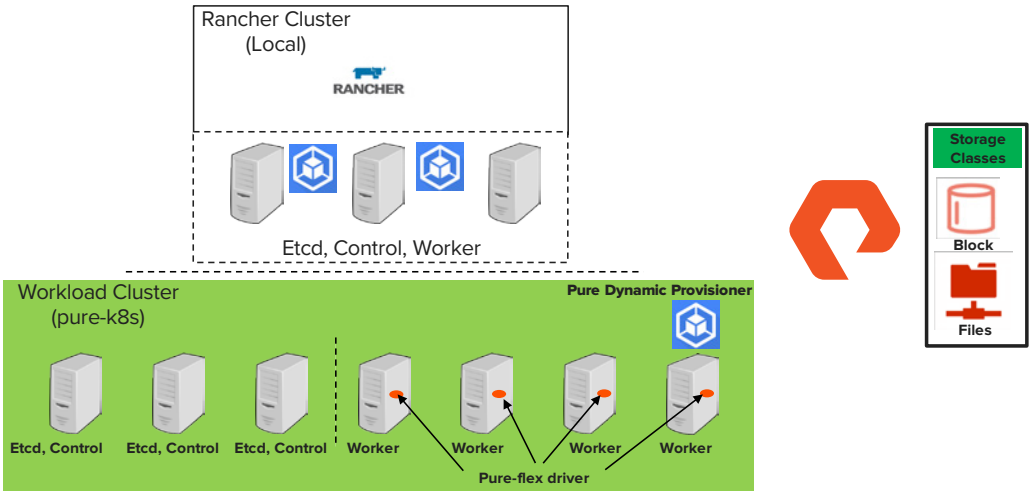

FIGURE 18. PSO layout in purek8s cluster

Using Helm charts is the preferred and easiest way to install PSO. Installing PSO using Rancher is further simplified. You can install PSO one time using the Rancher menu with the custom values.yaml file. Under **Catalog** in the **Global** menu, select **Add Catalog**.

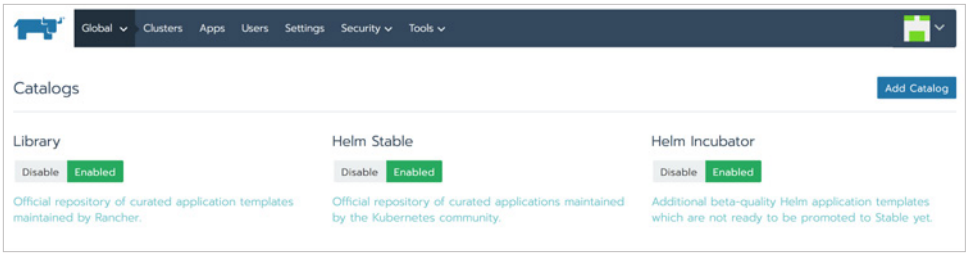

FIGURE 19. Catalog and stable application from Rancher Global menu

The **Add Catalog** will bring up a sub-menu that will require a name and the PSO's Helm chart location.

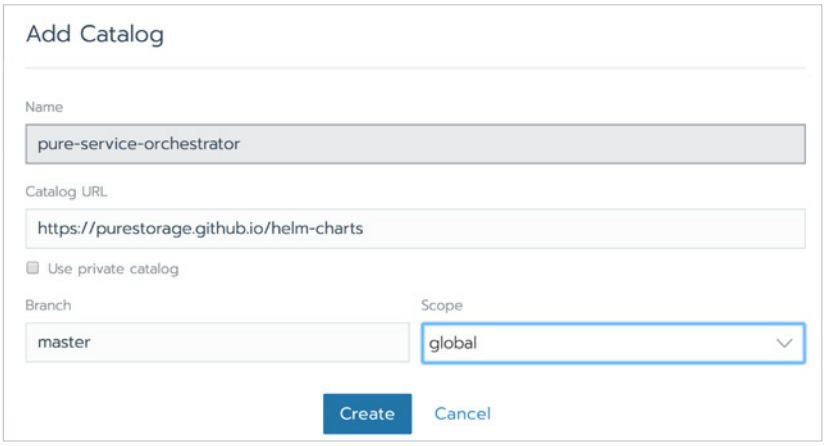

FIGURE 20. Adding the PSO Helm Chart to the Rancher Custom Catalog

Once the PSO is added to Catalog, install PSO in the new workload cluster created – **pure-k8s** under **system** for security reasons.

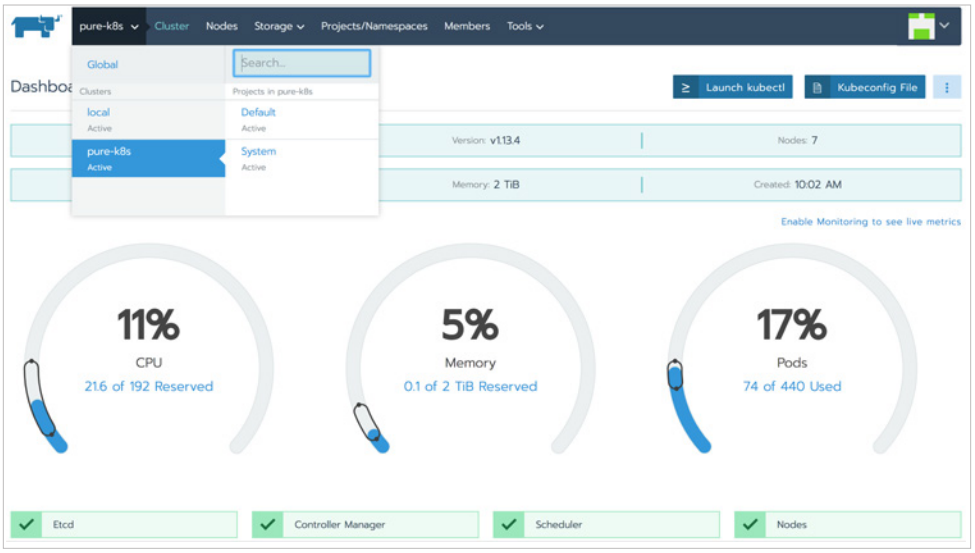

FIGURE 21. PSO installed under "system" namespace

The pure-k8s-plugin from **pure-service-orchestrator** will appear under the **Catalog Apps** in the Rancher menu under the pure-k8s cluster.

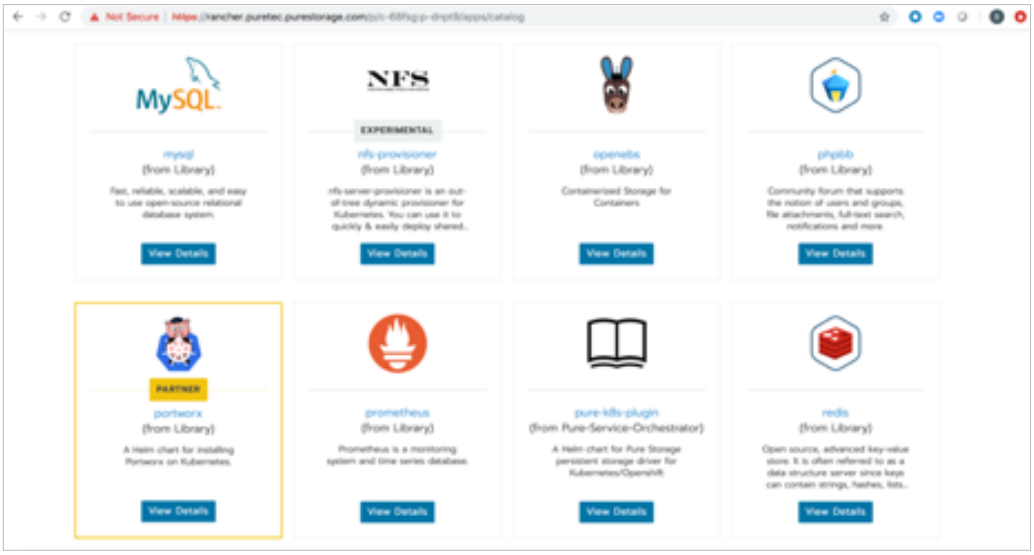

FIGURE 22. PSO listed as a Catalog Application

At the time of this writing, v2.2.1 was the latest PSO version available. Rancher allows you to upgrade to the latest version from its drop-down menu. The **View Details** link under **pure-k8s-plugin** leads to the next page, where you can paste the values.yaml file as **Edit as YAML**.

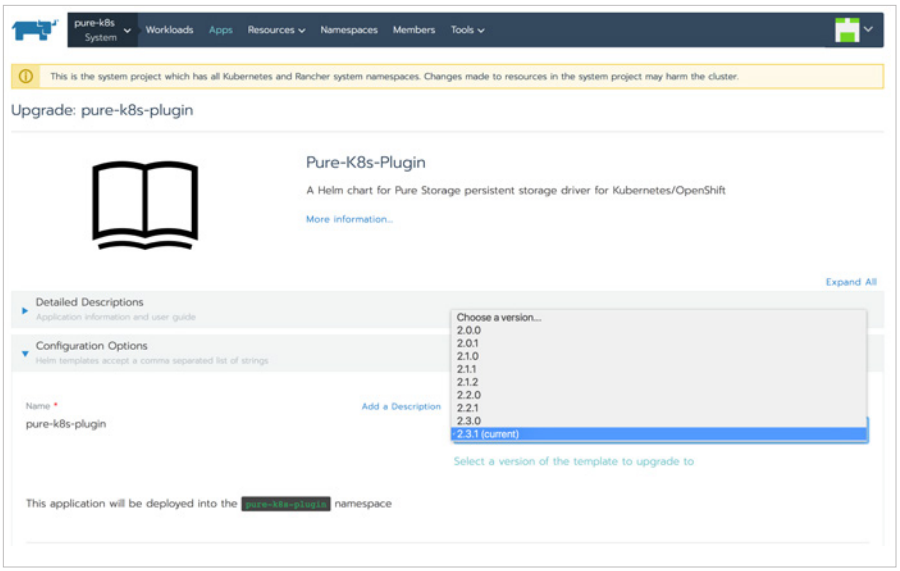

FIGURE 23. One-time PSO setup from Rancher

Prior to installing PSO, the following configurations are required on the Rancher management cluster (local) nodes and on the FlashArray:

1. Install **iscsi-initiator-util**s and **multipath** tools on all the Rancher management Cluster (local) and Workloads cluster (pure-k8s) nodes as a pre-requisite. In this example, there are three Rancher and seven pure-k8s 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 iSCISI volumes with a Rancher-launched Kubernetes cluster (pure-k8s), you need to add additional lines to the cluster yaml file. Edit the cluster **pure-k8s** to include the following lines under **services:**

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

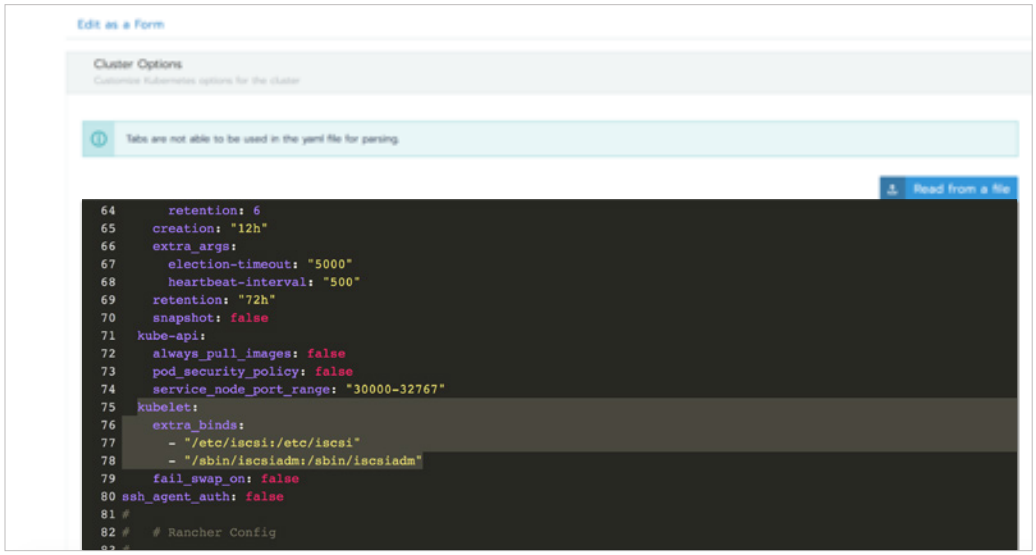

FIGURE 24. Additional settings for iSCSI volumes on FlashArray

> **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 the namespace, the path for the FLEX driver, and others.

```
# Default values for k8s-plugin.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

image:
  name: purestorage/k8s
  tag: 2.2.1
  pullPolicy: IfNotPresent

# this option is to enable/disable the debug mode of this app
# for pure-provisioner and pure-flex-daemon
app:
  debug: false

# do you want to set pure as the default storageclass?
storageclass:
  isPureDefault: false
  # set the type of backend you want for the 'pure' storageclass
  # pureBackend: file

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

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

# there are two namespaces for this app
# 1. namespace.pure is the backend storage namespace where volumes/shares/etc
```

```
#    will be created.
namespace:
  pure: pure-k8s-plugin

# support k8s or openshift
# if you want to install flex into a different place, you need to
# overwrite the flexpath.
orchestrator:
  # name is either 'k8s' or 'openshift'
  name: k8s

# flexPath is for image.tag >= 2.0
# `flexPath` needs to align with kubelet "volume-plugin-dir" configuration
# by default in Kubernetes it is '/usr/libexec/kubernetes/kubelet-plugins/volume/exec'
# Select an option below or customize for the environment.

# Default for Kubernetes and OpenShift on RHEL Server
flexPath: /usr/libexec/kubernetes/kubelet-plugins/volume/exec

# Default for Openshift 3.10+ on RHEL Atomic (containerized kubelet/origin-node)
#flexPath: /etc/origin/node/kubelet-plugins/volume/exec

# Default for Openshift 3.9 and lower with RHEL Atomic
#flexPath : /usr/libexec/kubernetes/kubelet-plugins/volume/exec

# Default for RKE
flexPath: /var/lib/kubelet/volumeplugins

# Default for GKE
#flexPath: /home/kubernetes/flexvolume    flexPath: /var/lib/kubelet/volumeplugins
# 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:
```

```
#       rack: "22"
#       env: "prod"
#   - MgmtEndPoint: "1.2.3.5"
#     APIToken: "b526a4c6-18b0-a8c9-1afa-3499293574bb"
 FlashBlades:
   - MgmtEndPoint: "10.21.241.11"
     APIToken: "T-1cf8cc93-0e34-4cce-9d91-d03b67c2c890"
     NfsEndPoint: "10.21.236.204"
#     Labels:
#       rack: "7b"
#       env: "dev"
#   - MgmtEndPoint: "1.2.3.8"
#     APIToken: "T-d4925090-c9bf-4033-8537-d24ee5669135"
#     NfsEndPoint: "1.2.3.9"
#     Labels:
#       rack: "6a
```

Once you've launched the **pure-k8s-plugin**, install and configure PSO with the settings listed in the values.yaml file. 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.

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 PSO from the Rancher menu, the Flex-driver should be running on all the worker nodes and the dynamic provisioner should be running as a pod in the pure-k8s cluster.

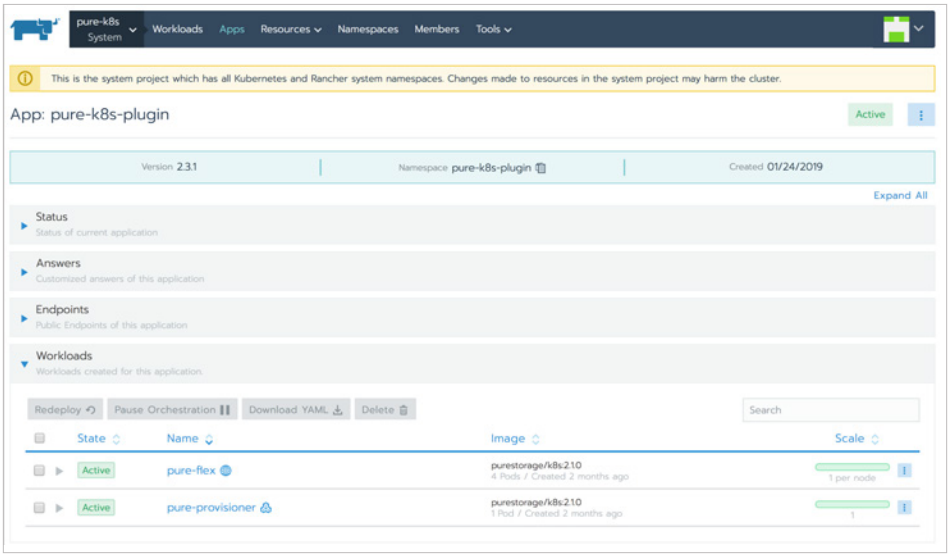

FIGURE 25. PSO Flex-driver running as "pod" in the pure-kk8s cluster

The pure-flex driver should be running on the worker nodes in the pure-k8s cluster:

```
[root@sn1-r720-g09-01 ~]# cd /var/lib/kubelet/volumeplugins/
[root@sn1-r720-g09-01 volumeplugins]# ls -al
total 12
drwxr-xr-x 3 root root 4096 Jan 24 10:26 .
drwxr-xr-x 8 root root 4096 Jan 14 12:51 ..
drwxr-xr-x 2 root root 4096 Feb 21 10:58 pure~flex
```

Now the Pure Storage classes are listed in the Rancher menu to be used by any application launched in the pure-k8s cluster.

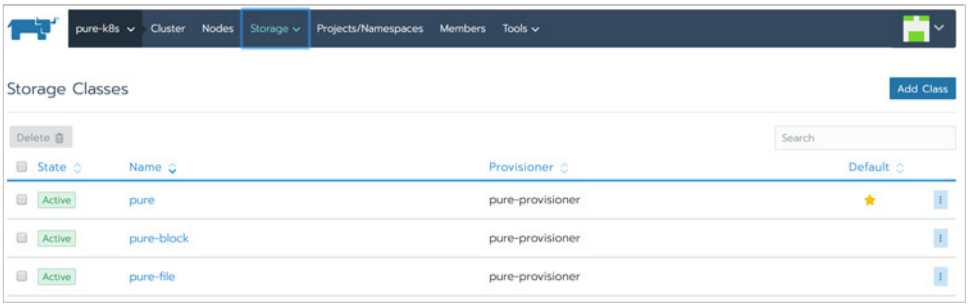

FIGURE 26. Pure Storage classes listed in Rancher

The storage classes available to the pure-k8s 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. PSO is now ready to dynamically manage and provision PVs on demand by users and by the application.

Before we start using PSO, 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 – 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 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-flex driver is responsible for attaching and mounting a PV based on the PVC's storage class.

Pure Service Orchestrator Workflow in a Kubernetes Cluster

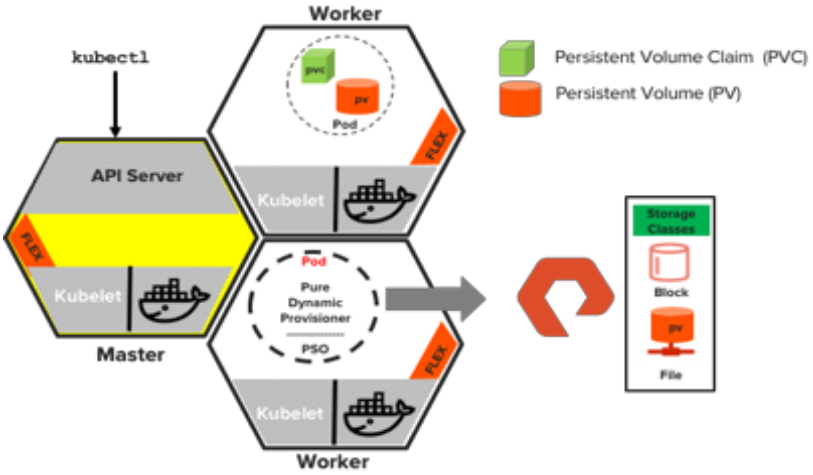

FIGURE 27. Pure dynamic provisioner and Kubernetes cluster

As depicted in Figure 27, 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 PSO to create the actual volume or file-system 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 is able to bind the PV to the PVC.

The following steps illustrate Pure FLEX driver's role in PSO.

1. The Kubernetes flex 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 flex volume.
3. Kubelet looks up the FLEX volume plugin associated with the PV.
4. Kubelet calls the Pure FLEX driver to mount the PV.
5. The Pure FLEX driver looks up the PV to find the array to attach. It finds FlashBlade.
6. The Pure FLEX 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 PSO — enables admins to accelerate Kubernetes environment readiness for developers and end-users, enabling seamless consumption of infrastructure. Production-ready installers allow admins to setup 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

As a Technical Director for DevOps/EDA, Bikash Roy Choudhury 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

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.

ps_wp36p_rancher-and-pure-service-orchestrator_01