Integrate Ceph and Kubernetes on Wiwynn ST7200-30P All-Flash Storage

Version 1.0

April

Copyright

Disclaimer

The information in this guide is subject to change without notice. Wiwynn Corporation makes no representations or warranties, either expressed or implied, with respect to the contents hereof and specifically disclaims any warranties of merchantability or fitness for any particular purpose. Any Wiwynn Corporation software described in this manual is sold or licensed "as is". Should the programs prove defective following their purchase, the buyer (and not Wiwynn Corporation, its distributor, or its dealer) assumes the entire cost of all necessary servicing, repair, and any incidental or consequential damages resulting from any defect in the software.

Revision History

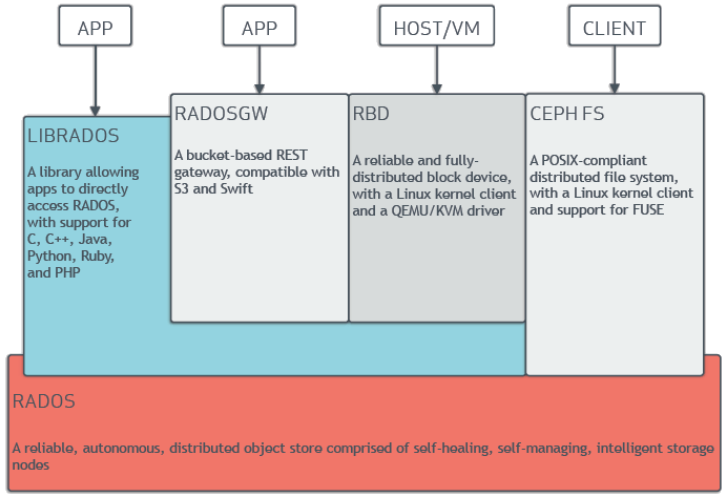
Date	Version	Changes
2018/04/09	1.0	First release

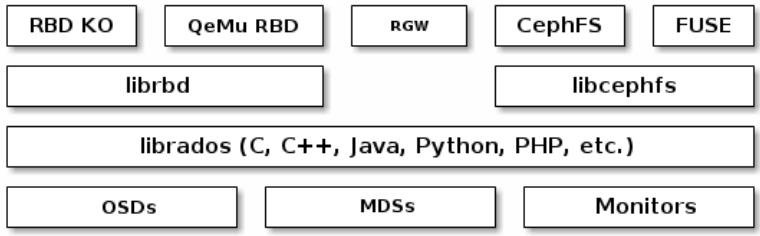
Contents

1. Introduction

Ceph

Ceph is a free software storage platform designed to present object, block, and file storage from a single distributed computer cluster. Ceph's main goals are to be completely distributed without a single point of failure, scalable to the Exabyte level, and freely-available. The data is replicated, making it fault tolerant. Ceph software runs on commodity hardware. The system is designed to be both self-healing and self-managing and strives to reduce both administrator and budget overhead.





Ceph OSDs

A Ceph OSD Daemon (Ceph OSD) stores data, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD Daemons for a heartbeat. A Ceph Storage Cluster requires at least two Ceph OSD Daemons to achieve an active + clean state when the cluster makes two copies of your data (Ceph makes 2 copies by default, but you can adjust it).

Monitors

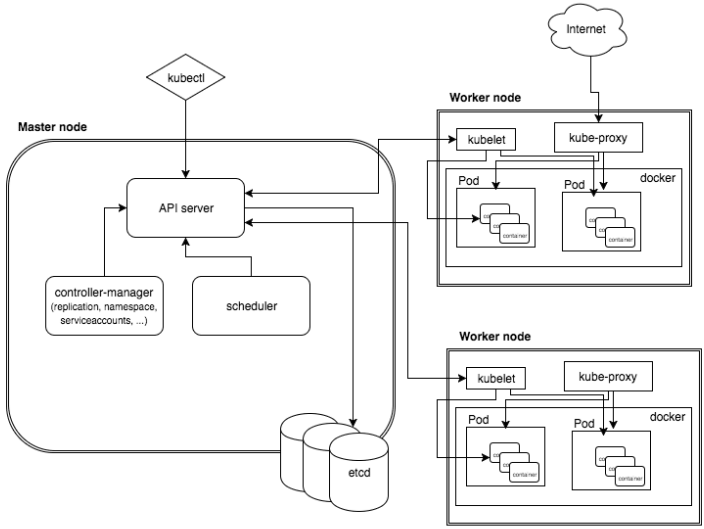
A Ceph Monitor maintains maps of the cluster state, including the monitor map, the OSD map, the Placement Group (PG) map, and the CRUSH map. Ceph maintains a history of each state change in the Ceph Monitors, Ceph OSD Daemons, and PGs.

MDSs

A Ceph Metadata Server (MDS) stores metadata on behalf of the Ceph Filesystem (i.e., Ceph Block Devices and Ceph Object Storage do not use MDS). Ceph Metadata Servers make it feasible for POSIX file system users to execute basic commands like ls, find, etc. without placing an enormous burden on the Ceph Storage Cluster.

Kubernetes

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes is built upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.



Let’s have a look into each of the component’s responsibilities.

Master Node

The master node is responsible for the management of Kubernetes cluster. This is the entry point of all administrative tasks. The master node orchestrates the worker nodes, where the actual services are running.

Let's dive into each of the components of the master node.

API Server

The API server is the entry point for all the REST commands used to control the cluster. It processes the REST requests, validates them, and executes the bound business logic. The result state has to be persisted somewhere, and that brings us to the next component of the master node.

etcd Storage

etcd is a simple, distributed, consistent key-value store. It is mainly used for shared configuration and service discovery. It provides a REST API for CRUD operations as well as an interface to register watchers on specific nodes, which enables a reliable way to notify the rest of the cluster about configuration changes.

An example of data stored by Kubernetes in etcd is jobs being scheduled, created and deployed, pod/service details and state, namespaces and replication information, etc.

Scheduler

The deployment of configured pods and services onto the nodes happens thanks to the scheduler component. The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence, is able to decide where to deploy a specific service.

Controller-manager

Optionally, you can run different kinds of controllers inside the master node. Controller-manager is a daemon that embeds the core control loops shipped with **Kubernetes**. A controller uses API-Server to watch the shared state of the cluster and makes corrective changes to the current state to change it to the desired one.

An example of such a controller is the Replication controller, which takes care of the number of pods in the system. The replication factor is configured by the user while the controller recreates a failed pod or removes an extra-scheduled one.

Other examples of controllers are endpoints controller, namespace controller, and service accounts controller which we will not dive into details here.

Worker Node

The pods are run here, so the worker node contains all the necessary services to manage networking between the containers, communicate with the master node, and assign resources to the containers scheduled.

Docker

Docker runs on each of the worker nodes, and runs the configured pods. It takes care of downloading the images and starting the containers.

Kubelet

kubelet gets the configuration of a pod from the API-Server and ensures that the described containers are up and running. This is the worker service that is responsible for communicating with the master node. It also communicates with etcd, to get information about services and write the details about newly created ones.

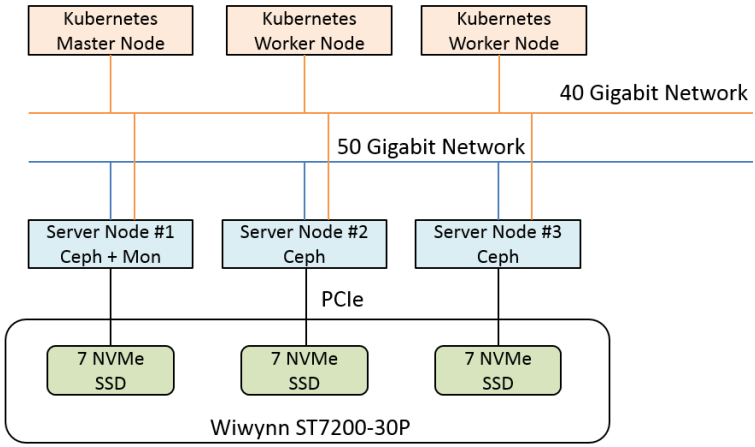
Kube-proxy

kube-proxy acts as a network proxy and a load balancer for a service on a single worker node. It takes care of the network routing for TCP and UDP packets.

Kubectl

And the final bit – a command line tool to communicate with the API service and send commands to the master node.

2. Testing Architecture



Ceph Reference Architecture	
JBOF Model	1 x Wiwynn ST7200-30P All-Flash Storage
NVMe slots	21 x 2.5 inch INTEL P3520 NVMe SSD
Server Model	6 x Wiwynn SV7221G2 OCP Server
CPU	2 x INTEL E5-2660V4
Memory slots	2 x 32GB
Network	Mellanox ConnectX-4 LX 50GbE for internal Mellanox ConnectX-3 40GbE for external
Version of Linux	CentOS Linux release 7.3.1611
Version of Ceph	10.2.9
Version of Kubernetes	1.7.5
Number of Ceph nodes	3
Number of Kubernetes nodes	3

3. Ceph Cluster Deployment

Ceph Deploy Setup

1. Put all host IPs in /etc/hosts for each ceph node and client node.

```
172.16.66.51    lpbw01os
172.16.66.52    lpbw02os
172.16.66.53    lpbw03os
```

```
yum install -y
https://dl.fedoraproject.org/pub/epel/epel-release-lates
t-7.noarch.rpm
```

2. Add a yum repository file for Ceph installation.

3. Add the Ceph repository to your yum configuration file at /etc/yum.repos.d/ceph-deploy.repo with the following command:

```
[ceph-noarch]
name=Ceph noarch packages
baseurl=https://download.ceph.com/rpm/el7/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://download.ceph.com/keys/release.asc
```

This will use the latest stable Ceph release. If you want to install a different release, replace “https://download.ceph.com/rpm/el7/noarch” with “https://download.ceph.com/rpm-{ceph-release}/el7/noarch” where {ceph-release} is a release name like luminous.

Install Ceph-deploy

```
sudo yum update
sudo yum install ceph-deploy
```

KERNEL Tuning

Kernel tuning must be performed on all ceph nodes.

Modify system control in /etc/sysctl.conf:

```
# Controls IP packet forwarding
net.ipv4.ip_forward = 0
# Controls source route verification
net.ipv4.conf.default.rp_filter = 1
# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0
# Controls the System Request debugging functionality of the
kernel
kernel.sysrq = 0
# Controls whether core dumps will append the PID to the core
filename.
# Useful for debugging multi-threaded applications.
kernel.core_uses_pid = 1
# disable TIME_WAIT.. wait ..
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
# Controls the use of TCP syncookies
net.ipv4.tcp_syncookies = 0
# double amount of allowed conntrack
net.netfilter.nf_conntrack_max = 2621440
net.netfilter.nf_conntrack_tcp_timeout_established = 1800
# Disable netfilter on bridges.
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
# Controls the maximum size of a message, in bytes
```

```
kernel.msgmnb = 65536
# Controls the default maxmimum size of a mesage queue
kernel.msgmax = 65536
# Controls the maximum shared segment size, in bytes
kernel.shmmax = 68719476736
# Controls the maximum number of shared memory segments, in
pages
kernel.shmall = 4294967296
```

Ceph Node Setup

We recommend installing NTP on Ceph nodes (especially on Ceph Monitor nodes) to prevent issues arising from clock drift.

1. Install NTP for each node.

```
# yum install ntp ntpdate ntp-doc
# systemctl start ntpd
# chkconfig ntpd on
```

The ceph-deploy utility must login to a Ceph node as a user that has passwordless sudo privileges, because it needs to install software and configuration files without prompting for passwords.

2. Create a user on each Ceph Node.

```
# useradd -d /home/cephadm -m cephadm
# passwd cephadm
```

3. For the user you added to each Ceph node, ensure that the user has sudo privileges.

```
# echo "cephadm ALL = (root) NOPASSWD:ALL" | sudo tee
/etc/sudoers.d/cephadm
# sudo chmod 0440 /etc/sudoers.d/cephadm
```

4. Enable PASSWORD-LESS SSH

Generate the SSH keys, but do not use sudo or the root user. Leave the passphrase empty.

```
# ssh-keygen
```

Copy the key to each Ceph Node, replacing {username} with the user name you created with Create a Ceph User.

```
# ssh-copy-id {username}@lpbw01os
# ssh-copy-id {username}@lpbw02os
# ssh-copy-id {username}@lpbw03os
```

5. OPEN REQUIRED PORTS or Disable Firewall

It is recommended to disable the firewall. To disable the firewall, run the following command as root:

```
# systemctl disable firewalld
# systemctl stop firewalld
```

6. Disable SELINUX.

```
# setenforce 0
```

To configure SELinux persistently, modify the configuration file at /etc/selinux/config.

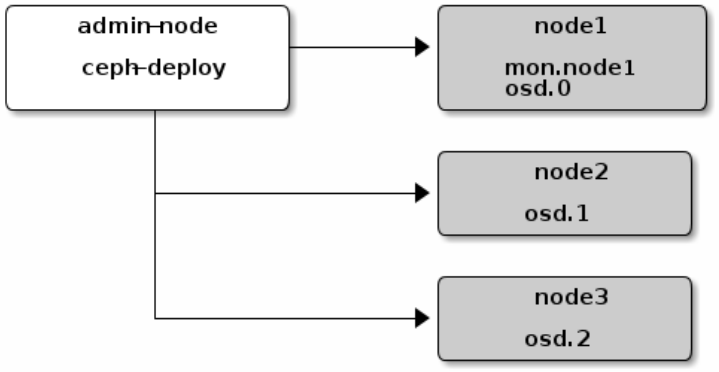
```
SELINUX=permissive
```

Ensure that your package manager has priority/preferences packages installed and enabled. On CentOS, you may need to install EPEL.

```
# yum install yum-plugin-priorities
```

Create Ceph Cluster

Create a three-Ceph-Node cluster so you can explore Ceph functionality.



Deploy Ceph

IMPORTANT! **If at any point you run into trouble and want to start over, execute the following commands to purge the configuration:**

```
# ceph-deploy purgedata lpbw01os lpbw02os lpbw03os
# ceph-deploy forgetkeys
# ceph-deploy purge lpbw01os lpbw02os lpbw03os
```

Create a temporary directory for installation. Create a directory on your admin node for maintaining the configuration files and keys that ceph-deploy generates for your cluster.

```
# su - cephadm
# mkdir my-cluster
# cd my-cluster
# ceph-deploy new lpbw01os
```

Install Ceph

1. Run this command.

```
# ceph-deploy install lpbw01os lpbw02os lpbw03os
```

2. Add the initial monitor and gather the keys.

```
# ceph-deploy mon create-initial
```

3. Once you complete the process, your local directory should have the following keyrings:

   - {cluster-name}.client.admin.keyring
   - {cluster-name}.bootstrap-osd.keyring
   - {cluster-name}.bootstrap-mds.keyring
   - {cluster-name}.bootstrap-rgw.keyring

   Use ceph-deploy to copy the configuration file and admin key to your admin node and your Ceph Nodes so that you can use the ceph CLI without having to specify the monitor address and ceph.client.admin.keyring each time you execute a command.

```
# ceph-deploy admin lpbw01os lpbw02os lpbw03os
```

4. Ensure that you have the correct permissions for the ceph.client.admin.keyring on each ceph node.

```
# chmod +r /etc/ceph/ceph.client.admin.keyring
```

5. Check your cluster's health.

```
# ceph health
```

Adding OSD

Check the disk listing on the ceph nodes for adding the Ceph cluster.

```
# ceph-deploy disk list {node-name [node-name]...}
```

ZAP Disks

To zap a disk (delete its partition table) in preparation for use with Ceph, execute the following:

```
# ceph-deploy disk zap lpbw01os:/dev/nvme{0..6}n1
# ceph-deploy disk zap lpbw02os:/dev/nvme{0..6}n1
# ceph-deploy disk zap lpbw03os:/dev/nvme{0..6}n1
```

Create OSDS

You may prepare OSDs, deploy them to the OSD node(s) and activate them in one step with the create command. The create command is a convenience method for executing the “prepare” and “activate” commands sequentially.

```
# ceph-deploy osd create lpbw01os:/dev/nvme{0..6}n1
# ceph-deploy osd create lpbw02os:/dev/nvme{0..6}n1
# ceph-deploy osd create lpbw03os:/dev/nvme{0..6}n1
```

Check OSD

```
# ceph osd tree
```

4. Kubernetes Cluster Deployment

System Configuration

Before Kubernetes installation, add some records in /etc/hosts for the Kubernetes nodes.

```
172.16.66.61    client01
172.16.66.62    client02
```

Disable Firewall

To make it easier to install Kubernetes, it is recommended to disable the firewall function on all Kubernetes nodes.

```
# setenforce 0
# vi /etc/selinux/config
SELINUX=disabled

# systemctl disable firewalld
# systemctl stop firewalld
```

Modify Kernel Parameters

Create a kernel parameter configuration file -- /etc/sysctl.d/k8s.conf, and add some bridge modules.

```
# vi /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-iptables =1

# sysctl -p /etc/sysctl.d/k8s.conf
```

Install Docker

1. Add yum repository for docker installation.

```
# yum install -y yum-utils device-mapper-persistent-data
lvm2

# yum-config-manager \
  --add-repo \
  https://download.docker.com/linux/centos/docker-ce.repo

# yum makecache fast
```

2. Check the docker version.

```
# yum list docker-ce.x86_64 --showduplicates |sort -r
docker-ce.x86_64     17.09.0.ce-1.el7.centos
docker-ce-stable
docker-ce.x86_64     17.06.2.ce-1.el7.centos
docker-ce-stable
docker-ce.x86_64    17.06.1.ce-1.el7.centos    docker-ce-stable
docker-ce.x86_64    17.06.0.ce-1.el7.centos    docker-ce-stable
docker-ce.x86_64    17.03.2.ce-1.el7.centos    docker-ce-stable
```

3. Install the latest version docker 17.09.

```
# yum install docker-ce-17.09.0.ce-1.el7.centos
```

Modify Docker Bridge IP

The default IP address segment is 172.17.xx.xx in Docker setting. If you want to use another network segment instead of the default subnet, then you can add new record in /etc/docker/daemon.json,

```
{
       "bip": "192.168.168.1/24"
}
```

1. Start docker service.

```
# systemctl start docker
# systemctl enable docker
```

2. Run this command to allow docker communication across all Kubernetes nodes:

```
iptables -P FORWARD ACCEPT
```

Install Kubeadm/Kubelet

1. Install kubeadm and kubelet on the master nodes and the worker nodes.

```
# edit /etc/yum.repos.d/kubernetes.repo

[kubernetes]
name=Kubernetes
baseurl=http://yum.kubernetes.io/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
  https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

# yum makecache fast
```

2. Check the latest version of kubeadm, kubelet, kubectl and kubernets-cni.

```
# yum list kubeadm  --showduplicates |sort -r
# yum list kubelet --showduplicates |sort -r
# yum list kubectl --showduplicates |sort -r
# yum list kubernetes-cni --showduplicates |sort -r
```

3. Install kubeadm 1.7.5 and kubelet 1.7.5, kubernetes-cni 0.5.1-0.

```
# yum install -y kubelet kubeadm kubectl kubernetes-cni
# systemctl enable kubelet.service
```

Initial Kubernetes Cluster

1. Use the kubeadm utility to initiate Kubernetes cluster. Before that, please make sure Docker cgroup and kubelet config are the same.

```
# docker info |grep -i cgroup
# cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

2. If there are different configurations, change the kubelet configuration to use cgroupfs.

```
KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs
```

3. Select client01 as the Master Node and run this command:

```
# kubeadm init \
--kubernetes-version=v1.7.5 \
--pod-network-cidr=10.244.0.0/16 \
--apiserver-advertise-address=172.16.66.61

[kubeadm] WARNING: kubeadm is in beta, please do not use it for production clusters.
[init] Using Kubernetes version: v1.7.5
[init] Using Authorization modes: [Node RBAC]
…
…
…
Your Kubernetes master has initialized successfully!
You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  http://kubernetes.io/docs/admin/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join --token 32a84a.67ed01b87055a46f 172.16.66.61:6443
```

4. To start using your cluster, you need to run (as a regular user):

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

5. Check Kubernetes cluster status.

```
# kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

Install Flannel (Pod) Network

1. Download and modify the flannel configuration file.

```
# mkdir -p ~/k8s/
# wget
https://raw.githubusercontent.com/coreos/flannel/v0.8.0/Do
cumentation/kube-flannel-rbac.yml
# wget
https://raw.githubusercontent.com/coreos/flannel/v0.8.0/Do
cumentation/kube-flannel.yml
```

2. Modify parameter “--iface” in kube-flannel.yml

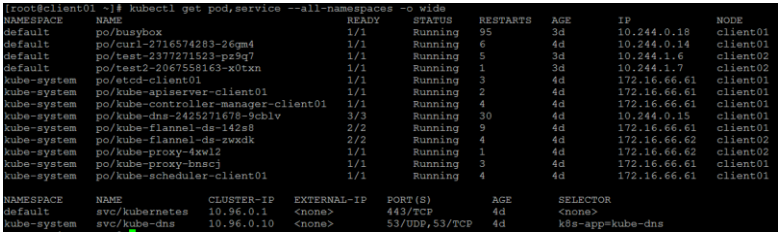
```
......
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: kube-flannel-ds
......
containers:
  - name: kube-flannel
  image: quay.io/coreos/flannel:v0.8.0-amd64
  command: [ "/opt/bin/flanneld", "--ip-masq",
"--kube-subnet-mgr", "--iface=ens2" ]
......
```

3. Install flannel network add-on.

```
# kubectl create -f kube-flannel-rbac.yml
# kubectl apply -f kube-flannel.yml
serviceaccount "flannel" created
configmap "kube-flannel-cfg" created
daemonset "kube-flannel-ds" created

# systemctl restart kubelet.service
```

4. Wait for 2 to 3 minutes for all Kubernetes services to start, then run the kubectl command to check the service status.



```
[root@client01 ~]# kubectl get pod,service --all-namespaces -o wide
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE     IP              NODE
default       po/busybox                                 1/1     Running   95         3d      10.244.0.18     client01
default       po/curl-2716574283-26gm4                   1/1     Running   6          4d      10.244.0.14     client01
default       po/test-2377271523-pz9q7                   1/1     Running   5          3d      10.244.1.6      client02
default       po/test2-2067558163-x0txn                  1/1     Running   1          3d      10.244.1.7      client02
kube-system   po/etcd-client01                           1/1     Running   3          4d      172.16.66.61    client01
kube-system   po/kube-apiserver-client01                 1/1     Running   2          4d      172.16.66.61    client01
kube-system   po/kube-controller-manager-client01        1/1     Running   4          4d      172.16.66.61    client01
kube-system   po/kube-dns-2425271678-9cblv               3/3     Running   30         4d      10.244.0.15     client01
kube-system   po/kube-flannel-ds-142s8                   2/2     Running   9          4d      172.16.66.61    client01
kube-system   po/kube-flannel-ds-zwxdk                   2/2     Running   4          4d      172.16.66.62    client02
kube-system   po/kube-proxy-4xwl2                        1/1     Running   1          4d      172.16.66.62    client02
kube-system   po/kube-proxy-bnscj                        1/1     Running   3          4d      172.16.66.61    client01
kube-system   po/kube-scheduler-client01                 1/1     Running   4          4d      172.16.66.61    client01

NAMESPACE     NAME             CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE       SELECTOR
default       svc/kubernetes   10.96.0.1    <none>        443/TCP         4d        <none>
kube-system   svc/kube-dns     10.96.0.10   <none>        53/UDP,53/TCP   4d        k8s-app=kube-dns
```

5. In Kubernetes, the master node places container workloads in user pods, on worker nodes or on the master node itself.

```
# kubectl get pod,svc --all-namespaces -o wide
```

6. But for testing purposes, we can change the Kubernetes setting to enable the Master node to take docker workload using this command:

```
# kubectl taint nodes client01 node-role.kubernetes.io/master-
node "client01" untainted
```

Join Your Nodes

The nodes are where your workloads (containers and pods, etc) run. To add new nodes to your cluster, perform the following for each machine:

1. SSH to the machine, then run the below command to join Kubernetes clusters.

```
# kubeadm join --token 32a84a.67ed01b87055a46f 172.16.66.61:6443
# kubeadm token list
```

2. This token can be found in this command below:

```
# kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/mas
ter/src/deploy/recommended/kubernetes-dashboard.yaml
```

Install Dashboard

```
# kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/re
commended/kubernetes-dashboard.yaml
```

5. Persistent Volume

In Kubernetes, containers are managed using deployments and they are termed as pods. Deployment holds the specification of pods. It is responsible to run the pod with specified resources. When pod is restarted or deployment is deleted, then data is lost on pod. We need to retain data out of pods lifecycle when the pod or deployment is destroyed.

A Persistent Volume (PV) in Kubernetes represents a real piece of underlying storage capacity in the infrastructure. In this paper, we use Ceph, an open source software defined storage, as the Persistent Storage for Kubernetes cluster.

Create Ceph-secret

1. Install ceph-common on Kubernetes nodes.

```
# yum install ceph-common
```

2. Query the secret value of client.admin from the Ceph node.

```
# ceph auth get-key client.admin
AQAtcoFZ3Ox5GRAAW0TxMCOPos6FiFTkcLktyg==
```

3. Convert to base64 code.

```
# echo "AQAtcoFZ3Ox5GRAAW0TxMCOPos6FiFTkcLktyg==" | base64
QVFBdGNvRlozT3g1R1JBQVcwVHhNQ09Qb3M2RmlGVGtjTGt0eWc9PQo=
```

4. Create ceph-secret.yaml.

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
data:
  key:
QVFBdGNvRlozT3g1R1JBQVcwVHhNQ09Qb3M2RmlGVGtjTGt0eWc9PQo=
```

Create a Secret in Kubernetes

```
# kubectl apply -f ceph-secret.yaml
# kubectl get secret
NAME          TYPE                                  DATA      AGE
ceph-secret   Opaque                                1         8s
default-token-6w73k  kubernetes.io/service-account-token  3     4d
```

Create rbd Image in Ceph Cluster

```
# rbd create test-image --pool data --image-format 1 --size 409600
rbd: image format 1 is deprecated

# rbd info data/test-image
rbd image 'test-image':
       size 400 GB in 102400 objects
       order 22 (4096 kB objects)
       block_name_prefix: rb.0.6a006.238e1f29
       format: 1
```

Create a PV in Kubernetes

1. Create test-pv.yaml.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test-pv
spec:
  capacity:
    storage: 400Gi
  accessModes:
    - ReadWriteOnce
  rbd:
    monitors:
      - 172.16.66.51:6789
    pool: data
    image: test-image
    user: admin
    secretRef:
      name: ceph-secret
    fsType: ext4
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

2. Create a new PV.

```
# kubectl apply -f test-pv.yaml
# kubectl get pv
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS
CLAIM     STORAGECLASS   REASON    AGE
test-pv   400Gi      RWO           Recycle         Available
2m
```

Create a PVC in Kubernetes

1. Create test-pvc.yaml.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 400Gi
```

2. Create a new PVC.

```
# kubectl apply -f test-pvc.yaml
# kubectl get pvc
NAME   STATUS    VOLUME  CAPACITY  ACCESSMODES  STORAGECLASS   AGE
test-claim    Bound    test-pv   400Gi     RWO              32s
```

Create a POD

1. Create nginx-deployment.yaml.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-dm
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
          volumeMounts:
            - name: ceph-rbd-volume
              mountPath: "/usr/share/nginx/html"
      volumes:
      - name: ceph-rbd-volume
        persistentVolumeClaim:
          claimName: test-claim
```

Create New POD

```
# kubectl apply -f nginx-deployment.yaml
# kubectl get pods

NAME                          READY     STATUS    RESTARTS   AGE
nginx-dm-4095233071-18nr2   1/1       Running   0          45m
```

Verification

1. Check disk usage in docker.

```
# kubectl exec -it nginx-dm-4095233071-18nr2 -- df -h
```

2. Touch a new file.

```
# kubectl exec -it nginx-dm-4095233071-18nr2 \
-- touch /usr/share/nginx/html/testing.html
```

3. Check if file exists.

```
# kubectl exec -it nginx-dm-4095233071-18nr2 \
  -- ls -lt /usr/share/nginx/html
```

4. Delete POD.

```
# kubectl delete pod nginx-dm-4095233071-18nr2
```

5. Re-create POD.

```
# kubectl apply -f nginx-deployment.yaml
```

6. Check if file exists.

```
# kubectl exec -it nginx-dm-4095233071-18nr2 \
  -- ls -lt /usr/share/nginx/html
```

6. Wiwynn ST7200-30P/60M

- **Ultra-high IOPS and Microsecond Grade Latency**

Comparing other storages with SAS and SATA SSD, Wiwynn ST7200-30P/60M delivers ultra-high performance with I/O throughput of at least 13.8 GB/s, IOPs of 2390K (4KB random read), and best ratio of IOPS per watt.

- **High Density Storage System with 60 SSDs**

With an impressive storage capacity of 60 NVMe SSDs in a 2U chassis, Wiwynn ST7200-30P/60M stands out from other JBOF storage systems as having the densest storage capacity in the industry.

- **Flexible Choice of Various SSDs Form Factors**

Wiwynn ST7200-30P/60M adopts PCIe 3.0 U.2 or M.2 NVMe SSD, giving data centers the best flexible configurations with 30 pieces of 2.5" (7mm) or 60 pieces of M.2 22110 or 2280 SSDs.

- **Tool-less and Modularized Design for Easy Upgrade and Maintenance**

The Wiwynn ST7200-30P/60M drawer-like innovative design allows users to upgrade or maintain SSDs, fans and expander board instantly without tools. In addition, its modularized design offers painless upgrade of PCIe expander board to new generation.

- **Hot-pluggable SSDs**

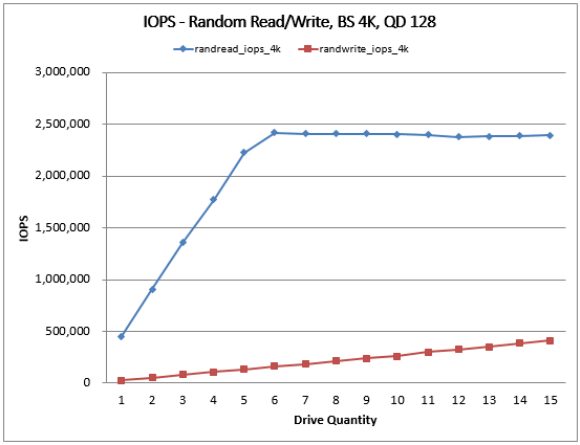
Without interrupting ongoing tasks or applications, hot-swap the SSDs on Wiwynn ST7200-30P/60M immediately when drive failures occur, thus, saving lots of time and efforts for maintenance and recovery.

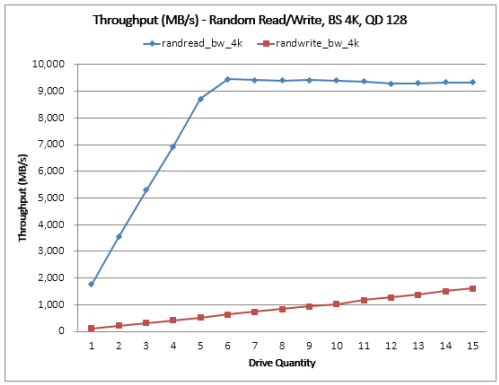
7. Performance

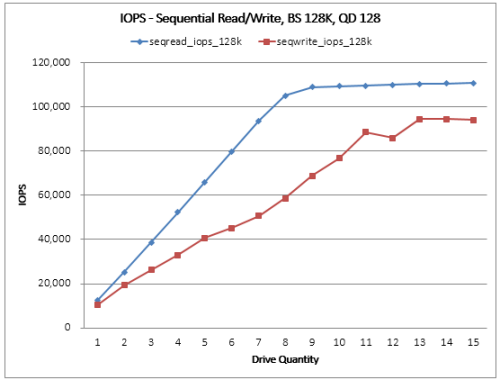
Raw Performance of Wiwynn ST7200-30P

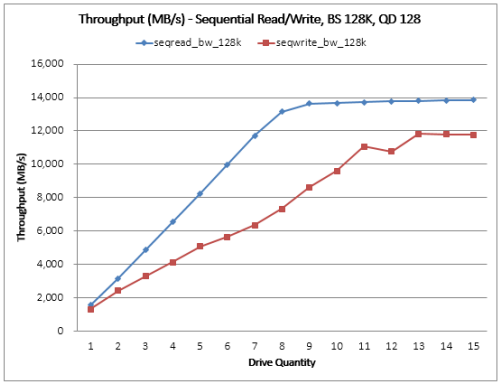
We had tested Wiwynn ST7200-30P JBOF, an all-flash storage by fio tools for raw data read and write performance. It can reach the maximum PCIe Gen3 x16 interface bandwidth of around 13GB when 7 to 8 NVMe SSDs are used in ST7200-30P storage. A maximum of 15 NVMe SSDs can be installed on this storage.

Based on specific NVMe SSDs tested on storage, the maximum random read IOPS is around 2390K for 4K block size and maximum sequential read throughput is around 13.8GB/s for 128K block size.




Throughput (MB/s) - Random Read/Write, BS 4K, QD 128
randread_bw_4k
randwrite_bw_4k
Throughput (MB/s)
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Drive Quantity


IOPS - Sequential Read/Write, BS 128K, QD 128
seqread_iops_128k
seqwrite_iops_128k
IOPS
120,000
100,000
80,000
60,000
40,000
20,000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Drive Quantity


Throughput (MB/s) - Sequential Read/Write, BS 128K, QD 128
seqread_bw_128k
seqwrite_bw_128k
Throughput (MB/s)
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Drive Quantity

Wiwynn ST7200-30P JBOF Performance

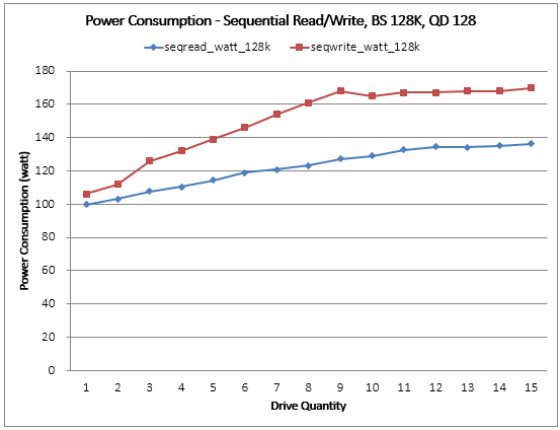
Type	IOPS	Throughput
4K Random Read	2,390,000	9.3GB/s
4K Random Write	410,000	1.6GB/s
128K Sequential Read	110,000	13.8GB/s
128K Sequential Write	94,000	11.7GB/s

** Queue Depth 128

Performance Optimization for Application

Storage Zoning	Performance	Application
Single Zone (15 SSDs)	Max. MB/s	Big Data, Streaming
Dual Zone (8 & 7 SSDs)	Max. IOPS	SQLDB, Web.

Wiwynn ST7200-30P JBOF with full load NVMe SSDs, maximum power consumption at around 180 watts.



Ceph Performance on Wiwynn ST7200-30P

In order to maximize Ceph performance, there are several layers to consider, including hardware, OS, file system, and Ceph layer. We focus on Wiwynn ST7200-30P JBOF and the real JBOF performance in Ceph environment. Therefore, we would turn off some cache management functions in OS and Ceph.

We collected RBD performance information from client nodes. Ceph admin node presented RBD images for client nodes access, then client nodes discovered these RBD images from remote and mounted it with EXT4.

For file system performance testing, we used fio tool to perform block size 4K random read and write, 128K sequential read and write, 1024K sequential read and write, to show up ST7200-30P JBOF all flash storage performance.

Two NVMe SSD for journal partitions for 5 OSDs

Block Size	Read	Write
4K Random	231K IOPS	46K IOPS
128K Sequential	9.4GB/s	2.6GB/s
1024K Sequential	11.4GB/s	3.3GB/s

- **Journal partition with each OSD (7 OSDs):** read performance was good, but write performance was not good, about 100MB/s.
- **1 NVMe SSD as journal for 6 OSDs: Good performance**
- **2 NVMe SSD as journal for 5 OSDs: Better performance**

Even NVMe SSD is faster than spinning disk, but one NVMe SSD as journal of write throughput is still not enough to support 6 OSDs to write data, the journal SSD of disk utilization says 89% during write performance testing. This journal SSD would be the performance bottleneck. Using two NVMe SSDs to support 5~6 OSDs significantly increased the write performance.

8. Summary

Containers and pods are ephemeral in Kubernetes, anything a container writes to its own file system gets wiped out when the container dies. Containers can also mount directories from their host node and read or write. That will survive container restarts, but the nodes themselves are not immortal.

There are other problems, such as ownership for mounted hosted directories when the container crashed. Just imagine a bunch of containers writing important data to various data directories on their hosts, and then crashed, leaving all data without direct way to tell which container the data was writing to. You can try to record this information, but where would you record it? It's pretty clear that for a large-scale system, you need persistent storage accessible from any node to reliably manage the data.

When using the NFS volume for Kubernetes, a file system (i.e. NFS) is mounted inside the pods. The file system allows multiple writes from different pods, which use the same persistent volume claim. Volumes can be shared between pods with the same data in each pod.

Benefits

Ceph is the only storage solution that delivers four critical capabilities:

- open-source
- software-defined
- enterprise-class
- unified storage (object, block, file).

Feature	Means	Final Benefit
Open Source	No license fees	Lower cost
Software-defined	Different hardware for different workloads Use commodity hardware Manage many nodes as one system	Broader use cases, higher efficiency. Lower cost, easier to evaluate. Easier to manage = lower operational cost.
Scale-out	Distributed capacity Distributed performance	Good performance from low cost servers
Block + Object	Store more types of data	Broader use cases
Enterprise features	Data protection Self-healing Data efficiency Caching/tiering	Don't lose valuable data Higher availability, easier management Lower cost Higher performance at lower cost

Many other storage products are open source, scale out, software-defined, unified or have enterprise features, but almost nothing else offers all four together.

Ceph also includes many basic enterprise storage features including: replication (or erasure coding), snapshots, thin provisioning, auto-tiering (ability to shift data between flash and hard drives), self-healing capabilities. Therefore, you can choose Ceph software storage solution in your enterprise.