Kubeadm Kubernetes Framework v0.9.0

Contents

Change Log:

Version	Date	Author	Changed
0.9.0	02/02/2025	Abiwot	Initial creation

About

Purpose

This document is intended to provide a comprehensive procedural guide on how to build a Kubernetes cluster framework to be the basis of a production level deployment.

Audience

This document is intended for a Development Operations (DevOps) level resources at an associate level knowledge of:

1. Linux
2. Kubernetes
   2.1. Networking
   2.2. Storage
   2.3. Role-based Access Control (RBAC)

How to navigate this document

You will notice there is multiple types of formatting within this document. Each has a significance that needs attention/modification while navigating.

Attention (critical)

1. Text **formatted in the following way** is to ensure you are notified of certain circumstances and/or conditions have been meet. This text is not to be modified and/or copied.

Attention (Important)

1. Text **formatted in the following way** or formatted in the following way is to ensure you are notified of certain circumstances and/or conditions are meet. This text is not to be modified and/or copied.

Code

1. Text **formatted in the following way** is to be copied/entered exactly the way presented. This text is not to be modified.
   1.1. e.g. **cp /etc/ntp.conf{,.$(date +%m%d%Y)}**
   1.2. e.g. **echo "autocmd FileType * setlocal formatoptions-=c formatoptions-=r formatoptions-=o" >> /etc/vimrc**

Variables

1. Any text encased with greater/less-than '**<>**'symbols, not in BOLD and highlighted is to be substituted with the appropriate data/text for the situation. This text is not to be copied into code/syntax inputs but changed to reflect the variable in the certain circumstance.
   1.1. e.g. **rabbitmqctl add_user vmwarerabbit** <password>
      1.1.1.e.g. **rabbitmqctl add_user vmwarerabbit pasword123**
   1.2. e.g. **less /storage/lvm_db01/data/log/postgresql-**<current day>**.log**
      1.2.1.e.g. **less /storage/lvm_db01/data/log/postgresql-Wed.log**

File Editing

1. When editing files within the operating system, you will notice a few different situations:
   1.1. Copy/modify exactly as seen
      1.1.1.e.g. **RPCNFSDARGS="-N2 -N3 -V4"**
   1.2. Modify with variable
      1.2.1.e.g. **#host all all** <IP network address> **md5**

1.2.2.e.g. **#host all all 10.254.80.101 md5**

Kubernetes Framework Component Matrix Compatibility

Component	Method	App ver	Helm ver
containerd (CRI)	APT	1.7.24	n/a
kubelet	APT	1.32.1-1.1	n/a
kubeadm	APT	1.32.1-1.1	n/a
kubectl	APT	1.32.1-1.1	n/a
kubernetes	manifest	1.32.1-1.1	n/a
calico (CNI)	helm	3.29.2	3.29.2
calicoctl	manifest	3.27.2	n/a
helm	manifest	3.17.0	n/a
Ingress-nginx (controller)	helm	1.12.0	4.12.0
rook-ceph	helm	1.16.1	1.16.1
kubernetes-csi-addon	manifest	0.11.0	n/a
k8 CSI external-Snapshotter	manifest	8.2.0	n/a
Kube-Prometheus-Stack (Alertmanager, Prometheus, Grafana)	helm	0.79.2	68.4.3
Kubernetes Dashboard	helm	7.10.3	7.10.3
cert-manager	helm	1.17.0	1.17.0

Prerequisites

See prerequisites.md within Github project ***https://github.com/abiwot/abiwot-kubeadm***

System requirements

See prerequisites.md within Github project ***https://github.com/abiwot/abiwot-kubeadm***

Building Initial Cluster

Prerequisites

Required OS Packages

These steps need to be duplicated on all nodes

General

1. Install general security packages
   1.1. Run **sudo apt update && sudo apt upgrade -y**
2. Install general packages that will be required for this process
   2.1. Run **sudo apt install -y vim curl wget git apt-transport-https ca-certificates gpg jq bash-completion**

Install YQ (YAMLquery)

1. Install YQ
   1.1. Run **sudo wget https://github.com/mikefarah/yq/releases/download/v4.44.6/yq_linux_amd64 -O /usr/bin/yq && sudo chmod +x /usr/bin/yq**

Disable local swap

(https://graspingtech.com/disable-swap-ubuntu/)

These steps need to be duplicated on all nodes

1. SSH into the first control plane node
2. Check if swap is on/enabled
   2.1. Run **sudo swapon --show**
   2.2. If swap is enabled, you will see similar:
   NAME TYPE SIZE USED PRIO
   /swap.img file 1.9G 4.5M -2
3. Permanently disable swap (if enabled)
   3.1. Run **sudo swapoff -a**
   3.2. Edit the FSTAB to comment out the swap
      3.2.1. Run **sudo sed -i.bak -r 's/(^\/swap.img)/#\1/' /etc/fstab**
   3.3. Run **sudo rm /swap.img**
   3.4. **Reboot system**

Installing Container Runtime Interface (CRI)

These steps need to be duplicated on all nodes

Containerd: Will be the default CRI

1. Load OS modules
   1.1. Configure modules to activate on boot
      1.1.1. Run **sudo modprobe overlay br_netfilter rbd nbd**
      1.1.2. Run

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
rbd
```

```
nbd
EOF
```

2. Modify SYSCTL parameters for CIS specific operations
    2.1. Run

```
cat <<EOF | sudo tee /etc/sysctl.d/75-kubelet-protect-kernel.conf
kernel.keys.root_maxbytes=25000000
kernel.keys.root_maxkeys=1000000
kernel.panic=10
kernel.panic_on_oops=1
vm.overcommit_memory=1
vm.panic_on_oom=0
EOF
```

3. Modify SYSCTL parameters for K8s general operations
    3.1. Run

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
net.netfilter.nf_conntrack_max = 1000000
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.core.netdev_max_backlog = 5000
fs.file-max = 500000
fs.inotify.max_user_instances = 8192
EOF
```

4. Reload any changes done to sysctl
    4.1. Run **sudo sysctl --system**
5. Modify OS open file limits
    5.1. Run **sudo sed -i -e '$a * soft nofile 65535\n* hard nofile 65535' /etc/security/limits.conf**
6. Install containerd
    6.1. Run **sudo apt-get update**
    6.2. Run **sudo apt install -y containerd=1.7.24***
7. APT hold/protect containerd package from upgrades
    7.1. Run **sudo apt-mark hold containerd**
8. Create containerd default configuration file
    8.1. Run **sudo mkdir -p /etc/containerd**
    8.2. Run **sudo containerd config default | sudo tee /etc/containerd/config.toml**
9. Modify containerd config
    9.1. Run **sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml**
10. Create containerd endpoints
    10.1. Run

sudo tee /etc/crictl.yaml <<EOF
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock
EOF

11. Run **sudo systemctl restart containerd**
12. Run **sudo systemctl enable containerd.service**

Install Kubernetes packages – kubeadm, kubelet, & kubectl

These steps need to be duplicated on all nodes

Google APT repository

1. Add Kubernetes APT repository
   1.1. Run **curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.32/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg**
   1.2. Run **echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.32/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list**

Install kubeadm, kubelet, & kubectl

1. Install packages
   1.1. Run **sudo apt update**
   1.2. To see available versions of Kubernetes available
      1.2.1. Run **apt-cache policy kubelet | head -n 20**
   1.3. Run **sudo apt install -y kubelet=1.32.1* kubeadm=1.32.1* kubectl=1.32.1***
2. APT hold/protect package upgrades
   2.1. Run **sudo apt-mark hold kubelet kubeadm kubectl**
3. Enable service boot time
   3.1. Run **sudo systemctl enable kubelet.service**

Clone K8s-framework Repository

You will want to clone the repo to all nodes. The repository will contain files required by control-plane, worker, and storage nodes.

These steps need to be duplicated on all nodes

1. Run **mkdir -p $HOME/projects/k8s/ && cd $HOME/projects/k8s/**
2. Clone K8s-framework tools
   2.1. Run **git clone --single-branch --branch main https://github.com/abiwot/abiwot-kubeadm.git**

CIS – Encryption-at-Rest

Enabling Kubernetes encryption for the etcd database is recommended. It is best to use a KMS system in a production environment. That is currently out-of-scope for this project.

We are going to use the local configuration file. It will provide encryption of etcd but lacks in securing from K8s host compromising (if the attacker gets access to your host, they will be able to get to your encryption-config file).

1. SSH into the initial control-plane
2. Create “enc” folder to maintain the encryption-config
    2.1. Run **sudo mkdir -p /etc/kubernetes/enc/**
3. Copy encryption-config to the kubernetes directory
    3.1. Run **sudo cp $HOME/projects/k8s/abiwot-kubeadm/framework/kubeadm/cluster-configs/control-plane/enc/encryption-config.yaml /etc/kubernetes/enc/**
4. Create a random key-values and inject into encryption-config
    4.1. **Make sure you copy the key-values into a secrets vault. You will need these later**
        4.1.1. Run **sudo sed -i "s|<key1-value>|$(head -c 32 /dev/urandom | base64)|" /etc/kubernetes/enc/encryption-config.yaml**
            4.1.1.1. Copy the value within **key1** in /etc/kubernetes/enc/encryption-config.yaml
5. Copy encryption-config to all control-plane nodes
    5.1. **ALL control-plane nodes MUST use the same encryption-config settings**
        5.1.1. Copy the above **/etc/kubernetes/enc/encryption-config.yaml** file to all control-plane nodes
6. Ensure you save the value (encryption key) of key1 to your password vault

Bootstrapping Cluster with kubeadm

Configure kubeadm Cluster Configuration

Prerequisites

1. Reboot all nodes
    1.1. To ensure any/all changes have populated through the OS
2. SSH into the 1st Kubernetes control-plane node
3. Create default clusterconfiguration.yaml
    3.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kubeadm/cluster-configs/control-plane**
4. Modify the **clusterconfiguration.yaml** to match your environment
    4.1. Variables
        4.1.1. advertiseAddress = IP of local control node
        4.1.2. kubernetesVersion = Version of installed packages in section “Install Kubernetes Packages”
    4.2. Set the node advertise address
        4.2.1. Run **sed -i 's/ advertiseAddress: .*/ advertiseAddress: <IP of localhost control-plane node>/' clusterconfiguration.yaml**
    4.3. Set Kubernetes version
        4.3.1. **It must match the Kubernetes version installed earlier in section “Install Kubernetes Packages”**
            e.g. *v1.32.1*
        4.3.2. Run **sed -i 's/^kubernetesVersion: .*/kubernetesVersion: <kubernetes packages installer earlier>/' clusterconfiguration.yaml**
    4.4. Set serviceSubnet configuration
        4.4.1. Recommend to setup/record each K8s cluster’s service and pod subnets
            4.4.1.1. Each K8s cluster must have unique internal IPs to ensure they do not overlap with each other.
        4.4.2. **The service subnet cannot overlap with the CNI or any host node network**
            e.g. periods(‘.’) and slashes(‘/’) must be escaped
                172\.16\.0\.0\/16
        4.4.3. Run **sed -i 's/^\ serviceSubnet:.*/\ serviceSubnet: <service subnet>/' clusterconfiguration.yaml**
    4.5. Set podSubnet configuration
        4.5.1. This pod subnet is the same pod subnet used in the Container Network Interface (CNI) settings. This will be the IP range your pods will be assigned.

e.g. periods('.') and slashes('/') must be escaped

172\.17\.0\.0\/16

4.5.2. Run **sed -i 's/^\ podSubnet: .*/\ podSubnet: <pod subnet>/' clusterconfiguration.yaml**

4.6. Set controller-manager CIDR allocation block sizes

4.6.1. This needs to match what will be set within the Calico Helm operator override values

4.6.2. This value is telling Kubernetes how to supernet the <podsubnet> for IPPool allocations

e.g 24 = supernet 172.17.0.0/16 into /24 networks

4.6.2.1.1. Run **sed -i 's|#<networking.podSubnet mask bits int32>|"<block size int32>"|' clusterconfiguration.yaml**

4.7. Set controlPlaneEndpoint configuration

controlPlaneEndpoint =

If using the external LB => FQDN of the external vIP or the vIP

If not using the external LB => FQDN of the localhost control plane node

4.7.1. Run **sed -i 's/^controlPlaneEndpoint: .*/controlPlaneEndpoint: <external LB vIP FQDN for API aggregation layer>/' clusterconfiguration.yaml**

4.7.1.1. If this is a single control plane node setup, then use the FQDN of the control plane node

e.g. cdak8clst100.abiwot-lab.com

Additional Initialization Files

These files are used to patch controllers and workers during the initialization or join phase.

Patches for Controllers

These steps need to be duplicated on all control-plane nodes

1. SSH into all controller node(s)
2. Source file location
   2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kubeadm/patches/controllers**
3. Run **sudo mkdir -p /etc/kubernetes/**
4. Copy the patches to **/etc/kubernetes/**
   4.1. Run **sudo cp *.yaml /etc/kubernetes/**

Patches for Workers & Storage Nodes

These steps need to be duplicated on all worker/storage nodes

1. SSH into worker node(s)
2. Source file location
   2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kubeadm/patches/workers**
3. Run **sudo mkdir -p /etc/kubernetes/patches/**
4. Copy patches to **/etc/kubernetes/patches/**
   4.1. Run **sudo cp *.json /etc/kubernetes/patches/**

Kubernetes Initial Cluster Initialization

1. SSH into the 1st Kubernetes control-plane node
2. Source file location
   2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kubeadm/cluster-configs/control-plane**
3. Generate bootstrap token for clusterconfiguration
   3.1. Run **kubeadm token generate**
      3.1.1. Capture output. You will require this information when joining the worker nodes
         3.1.1.1. **** Keep this information safe as it has access certificates ****
   3.2. Run **sed -i 's/<initBootstrapToken>.*/<token generated above>/' clusterconfiguration.yaml**

4. Run **sudo kubeadm init --config=clusterconfiguration.yaml --upload-certs**
   4.1. Copy the entire output to notepad or file; as this will be used later
      4.1.1. ** Keep this information safe as it has access certificates **
   4.2. You should see towards the end of the output a message similar to:
      Your Kubernetes control-plane has initialized successfully!
5. Configure local user to access Kubernetes
   5.1. Run **mkdir -p $HOME/.kube**
   5.2. Run **sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**
   5.3. Run **sudo chown $(id -u):$(id -g) $HOME/.kube/config**
6. Validate initial kube-system pods
   6.1. Validate running pods.
      6.1.1. Run **kubectl get pods -A**
         6.1.1.1. You should see similar output to:

NAMESPACE	*NAME*	*READY*	*STATUS*	*RESTARTS*
kube-system	*coredns-74ff55c5b-2s5z8*	*0/1*	*Pending*	*0*
kube-system	*coredns-74ff55c5b-5z6fx*	*0/1*	*Pending*	*0*
kube-system	*etcd-cdak8ctr001t*	*1/1*	*Running*	*0*
kube-system	*kube-apiserver-cdak8ctr001t*	*1/1*	*Running*	*0*
kube-system	*kube-controller-manager-cdak8ctr001*	*1/1*	*Running*	*0*
kube-system	*kube-proxy-t4cjp*	*1/1*	*Running*	*0*
kube-system	*kube-scheduler-cdak8ctr001t*	*1/1*	*Running*	*0*

You will notice the two 'coredns' pods are in a pending state. This will remain this way until the Container Network Interface (CNI) is installed and operational

Configure DNS Forwarding – Cluster Level

By default, the Kubernetes cluster will forward all unresolved DNS requests to '/etc/resolv.conf' locally on each host. To forward these request to a DNS server:

1. SSH into the control-plane node that was bootstrapped
2. Run **mkdir -p $HOME/projects/k8s/coredns && cd $HOME/projects/k8s/coredns**
3. Extract the CoreDNS configmap
   3.1. Run **kubectl get configmap -n kube-system coredns -o yaml > $HOME/projects/k8s/coredns/configmap-coredns.yaml**
4. Modify the configmap to add your DNS servers
   4.1. Ensure to change the IP addresses to reflect your DNS servers' IP address
      E.g. periods('.') and slashes('/') must be escaped
         DNS entries are space delimited
   4.2. Run **sed -i 's/forward . \/etc\/resolv\.conf {/forward . <DNSip DNSip> {/' $HOME/projects/k8s/coredns/configmap-coredns.yaml**
5. Apply the new configmap
   5.1. Run **kubectl apply -f $HOME/projects/k8s/coredns/configmap-coredns.yaml**

Adding Additional Controller Nodes

Join Controller nodes

1. Get the control-plane certificate-key & access token
   1.1. SSH into any initialized control plane node
   1.2. The output of the 'kubadm init' provides the certificate-key. If the --upload-certs flag was used, then the certificate will be available for 1 hour. After that period, the system will automatically remove the certificate.
      If within 1 hour of 'kubeadm init' => Use the certificate in the output
      If more than 1 hour, you will need to generate a new certificate-key
   1.3. Upload control-plane certificate key
      1.3.1. Run **sudo kubeadm init phase upload-certs --upload-certs | awk 'f;/Using certificate key:/{f=1}'**
         Copy the certificate and **keep secure**
   1.4. Generate control-plane access token
      1.4.1. Run **kubeadm token create --groups "system:bootstrappers:kubeadm:default-node-token" --ttl 20m --usages "authentication,signing" --description "Controller node addition" --print-join-command --certificate-key** <new key from init phase>
         Copy output and **keep secure**
            The access token in this command will only be **valid for 20 minutes**
2. Add the controller to the cluster
   2.1. **These steps will need to be duplicated on each new controller**
   2.2. SSH into the secondary controller (the node that was NOT already initialized).
   2.3. Join additional control-plane nodes
      2.3.1. The below command would have been part of the output of the first control-plane initialization process
         2.3.1.1 Run **sudo kubeadm join** <IP of external LB for K8 API>**:6443 --token** <access token> **--discovery-token-ca-cert-hash** <access token hash> **--control-plane --certificate-key** <control-plane certificate>
      2.3.2. You should receive similar output:

This node has joined the cluster and a new control plane instance was created:

** Certificate signing request was sent to apiserver and approval was received.*
** The Kubelet was informed of the new secure connection details.*
** Control plane (master) label and taint were applied to the new node.*
** The Kubernetes control plane instances scaled up.*
** A new etcd member was added to the local/stacked etcd cluster.*

To start administering your cluster from this node, you need to run the following 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
```

Run 'kubectl get nodes' to see this node join the cluster.

      5.1.1. If the join command seems to hang on the pre-flight checks, more than likely
         5.1.1.1. Incorrect access token, certificate, or hash
         5.1.1.2. Access token and/or certificate has expired
   5.2. Configure kubectl for local user on new controller
      5.2.1. Run **mkdir -p $HOME/.kube**
      5.2.2. Run **sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**
      5.2.3. Run **sudo chown $(id -u):$(id -g) $HOME/.kube/config**

5.3. Validate the new controller is ready

5.3.1. Run **kubectl get nodes**

5.3.1.1. You should see similar output:

NAME	STATUS	ROLES	AGE	VERSION
cdak8ctr001	*NotReady*	*control-plane,master*	*10m*	*v1.31.0*
cdak8ctr002	*NotReady*	*control-plane,master*	*5m2s*	*v1.31.0*

Validate Stacked kubeadm Cluster

Since this is a "stacked" kubeadm cluster, a few components need validation to ensure proper configurations.

REFERENCE: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/#stacked-etcd-topology

1. etcd - replication

1.1. Need to ensure all control-plane nodes are replicating the etcd database.

1.1.1. Run **kubectl exec -n kube-system etcd-<control-plane node name> -- etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key member list -w table**

1.1.2. You should see similar output:

```
+------------------+---------+-------------+---------------------------+---------------------------+------------+
|        ID        | STATUS  |    NAME     |        PEER ADDRS         |       CLIENT ADDRS        | IS LEARNER |
+------------------+---------+-------------+---------------------------+---------------------------+------------+
| 34278404ff4de941 | started | cdak8ctr002 | https://192.168.10.31:2380 | https://192.168.10.31:2379 |      false |
| 4ff11558f46d9f11 | started | cdak8ctr001 | https://192.168.10.32:2380 | https://192.168.10.32:2379 |      false |
| 800d311afea98f23 | started | cdak8ctr003 ! https://192.168.10.33:2380 | https://192.168.10.33:2379 |      false |
+------------------+---------+-------------+---------------------------+---------------------------+------------+
```

1.1.3. Column explanations

ID = unique identifier for each etcd node

STATUS =

started = member is up and running

unstarted = member is not operational

NAME = name of the node hosting etcd

PEER ADDRS = URL used by other etcd members to communicate with the local etcd member

CLIENT ADDRS = URL used by clients (kubectl) to interact with the local etcd member

IS LEARNER =

false = etcd member is full voting member

true = etcd member is syncing data but not yet participating in voting

2. etcd – encryption

2.1. View the raw data within the etcd

2.1.1. Run **kubectl exec -n kube-system etcd-<control-plane node name> -- etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key get /registry/secrets/kube-system/kubeadm-certs -w json | jq .**

2.2. Validate secret retrieval is encrypted

2.2.1. Run **kubectl get secrets -n kube-system kubeadm-certs -o=jsonpath='{.data.ca\.crt}' | base64 -d;echo**

2.2.1.1. The output should be encrypted.

2.2.2. If the output was able to be base64 decoded, then run the following command and repeat the test above

2.2.2.1. Run **kubectl get secrets --all-namespaces -o json | kubectl replace -f -**

Add Worker Nodes to Cluster

Join Worker & Storage Nodes

These steps need to be duplicated on all worker & storage nodes

1. Source file location
   1.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kubeadm/cluster-configs/workers**
2. Configure the join configuration for your environment
   2.1. Run **sed -i 's/^\   apiServerEndpoint:.*/\   apiServerEndpoint:** < external LB vIP FQDN for API aggregation layer at port 6443 >**/' worker-join-configuration.yaml**
   2.2. Run **sed -i 's/<initBootstrapToken>.*/**<token generated above>**/' worker-join-configuration.yaml**
      2.2.1. The **<token generated above>** = the token output generated when the initial control-plane node was initialized
3. Join worker node to cluster
   3.1. SSH into worker node
   3.2. Run **sudo kubeadm join** < external LB vIP FQDN for API aggregation layer at port 6443 > **--config worker-join-configuration.yaml**
      3.2.1. You should receive similar output:

[preflight] Running pre-flight checks

[preflight] Reading configuration from the cluster...

[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'

W0617 12:29:02.183349   28372 utils.go:69] The recommended value for "resolvConf" in "KubeletConfiguration" is: /run/systemd/resolve/resolv.conf; the provided value is: /run/systemd/resolve/resolv.conf

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Starting the kubelet

[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:

** Certificate signing request was sent to apiserver and a response was received.*

** The Kubelet was informed of the new secure connection details.*

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

4. Add label to **worker** nodes
   4.1. SSH into any controller node
   4.2. Run **kubectl label node** <worker node name> **node-role.kubernetes.io/worker=true**
5. Add label to **storage** nodes
   5.1. SSH into any controller node
   5.2. Run **kubectl label node** <storage node name> **node-role.kubernetes.io/storage=true**

6. Validate worker and storage nodes joined the cluster
    6.1. SSH into any control plane node
    6.2. Run **kubectl get nodes**
        6.2.1. You should receive similar output:

NAME	*STATUS*	*ROLES*	*VERSION*
cdak8ctr001	*NotReady*	*control-plane*	*v1.32.1*
cdak8ctr002	*NotReady*	*control-plane*	*v1.32.1*
cdak8ctr003	*NotReady*	*control-plane*	*v1.32.1*
cdak8str001	*NotReady*	*storage*	*v1.32.1*
cdak8str002	*NotReady*	*storage*	*v1.32.1*
cdak8str003	*NotReady*	*storage*	*v1.32.1*
cdak8wkr001	*NotReady*	*worker*	*v1.32.1*
cdak8wkr002	*NotReady*	*worker*	*v1.32.1*
cdak8wkr003	*NotReady*	*worker*	*v1.32.1*
cdak8wkr004	*NotReady*	*worker*	*v1.32.1*

If the worker node status is "NotReady", check every 10-20 secs

The worker node will not be ready until system pods (like Calico CNI) have completed deployment to the new worker node

etcdutl Environment Setup

At the time of writing this setup, the **etcdctl** is deprecated for certain etcd controls (but still functional) in favour of the **etcdutl** commands.

etcdctl / etcdutl Binaries

This process need to be completed on all control-plane nodes

1. SSH into control-plane node
2. Capture the etcd version
    2.1. Run **export ETCD_VER=v$(kubectl exec -it -n kube-system etcd-**<control-plane node name> **-- etcdctl version | awk 'NR==1 {print $NF}' | tr -d '\r')**
3. Set download source
    3.1. Run **export GITHUB_URL="https://github.com/etcd-io/etcd/releases/download" && export DOWNLOAD_URL=${GITHUB_URL}**
4. Remove any existing files in /tmp and create a directory
    4.1. Run **rm -f /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz**
    4.2. Run **rm -rf /tmp/etcd-download-test && mkdir -p /tmp/etcd-download-test**
5. Download and extract binaries
    5.1. Run **curl -L ${DOWNLOAD_URL}/${ETCD_VER}/etcd-${ETCD_VER}-linux-amd64.tar.gz -o /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz**
    5.2. Run **tar xzvf /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz -C /tmp/etcd-download-test --strip-components=1**
    5.3. Run **rm -f /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz**
6. Move binaries into /usr/local/bin
    6.1. Run **sudo mv /tmp/etcd-download-test/etcdctl /usr/local/bin/**
    6.2. Run **sudo mv /tmp/etcd-download-test/etcdutl /usr/local/bin/**
7. Verify version
    7.1. Run **etcdctl version**
    7.2. Run **etcdutl version**

Verify ETCD Connectivity

1. SSH into control-plane node
2. Run **export ETCDCTL_API=3**
3. Set an ENV of the ETCD members
   3.1. Run **export export ETCD_CLIENT_ADDRS=$(sudo etcdctl --endpoints=https://127.0.0.1:2379 --cert=/etc/kubernetes/pki/etcd/server.crt --cacert=/etc/kubernetes/pki/etcd/ca.crt --key=/etc/kubernetes/pki/etcd/server.key member list | awk -F', ' '{print $5}' | paste -sd, -)**
4. Get the status of each ETCD member
   4.1. Run **sudo etcdctl --endpoints=${ETCD_CLIENT_ADDRS} --cert=/etc/kubernetes/pki/etcd/server.crt --cacert=/etc/kubernetes/pki/etcd/ca.crt --key=/etc/kubernetes/pki/etcd/server.key endpoint status -w table**
      4.1.1. You should receive similar output:

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	IS LEARNER	RAFT TERM	RAFT INDEX	RAFT APPLIED INDEX	ERRORS
https://192.168.10.31:2379	34278404ff4de941	3.5.16	2.6 MB	false	false	2	5812	5812	
https://192.168.10.32:2379	4ff11558f46d9f11	3.5.16	2.7 MB	true	false	2	5812	5812	
https://192.168.10.33:2379	800d311afea98f23	3.5.16	2.6 MB	false	false	2	5812	5812	

Verify ETCD Snapshot Creation

1. SSH into control-plane node
2. Run **export ETCDCTL_API=3**
3. Create a snapshot of ETCD
   3.1. Run **sudo etcdctl --endpoints=https://127.0.0.1:2379 --cert=/etc/kubernetes/pki/etcd/server.crt --cacert=/etc/kubernetes/pki/etcd/ca.crt --key=/etc/kubernetes/pki/etcd/server.key snapshot save $HOME/snapshot_$(date +%m%d%Y%H%M%S).db**
   3.2. The last line of STDOUT will provide the exact location of the snapshot file
4. Validate snapshot
   4.1. Run **sudo etcdutl --write-out=table snapshot status $HOME/**<snapshot filename>

kubectl Environment Setup

Bash autocomplete for kubectl

This process needs to be completed on all control-plane nodes

1. Install and configure Bash autocomplete for kubectl
   1.1. SSH into all kubernetes control-plane node
      1.1.1. This process should be completed on all control plane nodes
   1.2. Run **sudo apt install -y bash-completion**
   1.3. Run **echo "source <(kubectl completion bash)" >> $HOME/.bashrc**
   1.4. Run **source $HOME/.bashrc**
2. Verify kubectl autocomplete is working
   2.1. Run **kubectl get po**<tab> **-n kube-**<tab>
      2.1.1. Each time you press the <tab> key, the autocomplete should finish off the syntax.
         2.1.1.1. You can also double press <tab> and get options for completion

Helm Install

These processes must be completed on all control-plane nodes

Installation

1. SSH into all control-plane nodes
    1.1. This process should be completed on all control plane nodes
2. Run **sudo snap install helm --classic**
3. Run **sudo snap refresh helm --hold**

Helm Repo Chart Initialize

1. SSH into all control plane nodes
    1.1. This process should be completed on all control plane nodes
2. Run **helm repo add k8s-dashboard https://kubernetes.github.io/dashboard**
    2.1. Raw command
        2.1.1. **helm repo add** <repo name> <repo URL>
3. Run **helm repo update**

Helm Diff Plugin

1. SSH into all control-plane nodes
    1.1. This process should be completed on all control-plane nodes
2. Run **helm plugin install https://github.com/databus23/helm-diff**

Helm Auto-completion BASH

1. SSH into all control-plane nodes
    1.1. This process should be completed on all control-plane nodes
2. Run **helm completion bash | sudo tee /etc/bash_completion.d/helm**
    2.1. This will be active on all new sessions. You will need to end your current session for the Helm auto-complete to take effect

Configure Network Overlay Environment (Calico)

Prerequisites

Remove Calico Interfaces from NetworkManger Control

This process needs to be completed on ALL nodes

Since NetworkManager is not the default network controller in Ubuntu, we should not need this but, will place it there as a precaution.

1. SSH into all nodes
2. Run **sudo mkdir -p /etc/NetworkManager/conf.d/**
3. Run

```
cat <<EOF | sudo tee /etc/NetworkManager/conf.d/calico.conf
[keyfile]
unmanaged-devices=interface-name:cali*;interface-name:tunl*;interface-name:vxlan.calico;interface-name:wireguard.cali
EOF
```

Calico – Operator Installation (Helm)

1. SSH into a control-plane node
2. Add the Tigera helm repository
   2.1. Run **helm repo add projectcalico https://docs.tigera.io/calico/charts**
   2.2. Run **helm repo update**
3. Modify the Helm values
   3.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/tigera-calico/helm**
   3.2. This pod subnet is the same pod subnet used in the Container Network Interface (CNI) settings. This will be the IP range your pods will be assigned.
      3.2.1. You can validate the pod network assigned by
         3.2.1.1. Run **kubectl cluster-info dump | grep -m 1 -- '--cluster-cidr'**
         e.g. periods('.') and slashes('/') must be escaped
         172\.17\.0\.0\/16
      3.2.2. Run **sed -i 's/cidr: <pod subnet>/cidr: "<pod subnet>"/' helm-calico-operator-override-values.yaml**
   3.3. Set the IPPool block allocation size. This needs to match what was configured in the kubernetes clusterconfiguration.yaml file
         e.g 24 = supernet 172.17.0.0/16 into /24 networks
      3.3.1. Run **sed -i 's/blockSize: <pod subnet mask bits>/blockSize: <pod subnet mask bit int32>/' helm-calico-operator-override-values.yaml**
   3.4. Disable BGP inter-node mechanism
      3.4.1. Since the future plan is to link multiple clusters with Submariner, we want VXLAN to be the encapsulation without BGP. The Calico default mechanism is BGP, see notes within file
      3.4.2. Run **sed -i 's/bgp: <Disabled | Enabled>/bgp: Disabled/' helm-calico-operator-override-values.yaml**
      3.4.3. Run **sed -i 's/encapsulation: <VXLAN>/encapsulation: VXLAN/' helm-calico-operator-override-values.yaml**
4. Install the operator
   4.1. Run **helm upgrade -i calico projectcalico/tigera-operator --version v3.29.2 --namespace tigera-operator --create-namespace -f helm-calico-operator-override-values.yaml**
5. Verify the operator
   5.1. Run **kubectl wait --namespace tigera-operator --for=condition=ready pod --selector=k8s-app=tigera-operator --timeout=120s**
6. Verify Calico
   6.1. Run **kubectl get po -n calico-system**
      6.1.1. You should receive similar output:

```
NAME                                     READY   STATUS    RESTARTS   AGE
calico-kube-controllers-795b645fdd-zhff6 1/1     Running   0          12m
calico-node-4c696                        1/1     Running   0          12m
calico-node-hjpdn                        1/1     Running   0          12m
calico-node-hnmvz                        1/1     Running   0          12m
calico-node-kdk72                        1/1     Running   0          12m
calico-node-kztx7                        1/1     Running   0          12m
calico-node-rg2t8                        1/1     Running   0          12m
calico-node-spt5t                        1/1     Running   0          12m
calico-node-ts697                        1/1     Running   0          12m
calico-node-wrtsd                        1/1     Running   0          12m
calico-typha-89bf456cc-4rvdv             1/1     Running   0          12m
calico-typha-89bf456cc-gczdp             1/1     Running   0          12m
calico-typha-89bf456cc-kwj4q             1/1     Running   0          12m
csi-node-driver-2fllc                    2/2     Running   0          12m
```

```
csi-node-driver-2h2wm      2/2   Running   0     12m
csi-node-driver-b8ndn      2/2   Running   0     12m
csi-node-driver-f99zj      2/2   Running   0     12m
csi-node-driver-pt25m      2/2   Running   0     12m
csi-node-driver-qz8v5      2/2   Running   0     12m
csi-node-driver-rcst7      2/2   Running   0     12m
csi-node-driver-xf5tp      2/2   Running   0     12m
csi-node-driver-zbccr      2/2   Running   0     12m
```

6.2. Run **kubectl get po -n calico-apiserver**

6.2.1. You should receive similar output:

```
NAME                                READY   STATUS
calico-apiserver-5fbcc84c97-6xvz6   1/1     Running
calico-apiserver-5fbcc84c97-bpsx5   1/1     Running
```

6.3. Run **kubectl get po -n kube-system -l k8s-app=kube-dns**

6.3.1. You should receive similar output:

```
NAME                       READY   STATUS    RESTARTS   AGE
coredns-5dd5756b68-62qzt   1/1     Running   0          39m
coredns-5dd5756b68-wcb4f   1/1     Running   0          39m
```

6.3.2. Notice the 'coredns' pods are now in a ready state (compared to previously installing the CNI)

6.4. Run **kubectl get ippools default-ipv4-ippool -o jsonpath='{.spec}' | jq**

6.4.1. You should receive similar output:


```
{
  "allowedUses": [
    "Workload",
    "Tunnel"
  ],
  "blockSize": 24,
  "cidr": "172.17.0.0/16",
  "ipipMode": "Never",
  "natOutgoing": true,
  "nodeSelector": "all()",
  "vxlanMode": "Always"
}
```


7. Verify the correct IPPool block sizes were assigned

7.1. Run **kubectl get nodes -o custom-columns="NODE:.metadata.name,POD_CIDR:.spec.podCIDR"**

Calico Configurations

Configure Calico for Prometheus monitoring

See "Install and Configure Prometheus"

Install & Configure calicoctl

Validate Calico API Server Operational

1. SSH into any control plane node

2. Run **kubectl get tigerastatus apiserver**
    2.1. You should receive similar output:

NAME	*AVAILABLE*	*PROGRESSING*	*DEGRADED*	*SINCE*
apiserver	***True***	***False***	***False***	***8d***

Install calicoctl

This process needs to be duplicated on all control-plane nodes

1. SSH into all control-plane nodes
    1.1. This should be completed on all control-plane nodes
2. Verify the Calico API server version
    2.1. Run **CALICOAPI0=$(kubectl get pods -n calico-apiserver -o jsonpath='{ .items[0].metadata.name}')**
    2.2. Run **kubectl get pods -n calico-apiserver $CALICOAPI0 -o jsonpath='{ .spec.containers[].image}{"\n"}'**
    2.3. Take note of the Calico image version. **The calicoctl version MUST match the API version**.
3. Install calicoctl as a plugin
    3.1. Run **mkdir -p $HOME/projects/k8s/tigera-calico/calicoctl/v3.29.2/ && cd $HOME/projects/k8s/tigera-calico/calicoctl/v3.29.2**
    3.2. Run **curl -L https://github.com/projectcalico/calico/releases/download/v**<calico API server version>**/calicoctl-linux-amd64 -o kubectl-calico**
    3.3. Run **chmod +x $HOME/projects/k8s/tigera-calico/calicoctl/v3.29.2/kubectl-calico**
    3.4. Run **sudo mv $HOME/projects/k8s/tigera-calico/calicoctl/v3.29.2/kubectl-calico /usr/local/bin**
    3.5. Run **kubectl calico ipam show**
        3.5.1. You should receive similar output:

GROUPING	*CIDR*	*IPS TOTAL*	*IPS IN USE*	*IPS FREE*
IP Pool	***171.17.0.0/16***	***65536***	***62 (0%)***	***962 (99%)***

Install and Configure Cert-Manager

Install Cert-Manager (CM)

1. SSH into a control-plane node
2. Add Helm repo
    2.1. Run **helm repo add jetstack https://charts.jetstack.io**
    2.2. Run **helm repo update**
3. Install Cert-Manager
    3.1. Install Cert-Manager CRDs
        3.1.1. Run **kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.17.0/cert-manager.crds.yaml**
    3.2. Install Cert-Manager
        3.2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/cert-manager/helm/**
        3.2.2. Run **helm upgrade -i cert-manager jetstack/cert-manager -n cert-manager --create-namespace --version v1.17.0 -f helm-cert-manager-override-values.yaml**
4. Verify cert-manager pods deployed
    4.1. Run **kubectl get pods -n cert-manager**

4.1.1. You should receive similar output:

NAME	READY	STATUS	RESTARTS
cert-manager-85f687f745-f8kdc	*1/1*	*Running*	*0*
cert-manager-85f687f745-wc5w6	*1/1*	*Running*	*0*
cert-manager-cainjector-784f978fb7-h2sqb	*1/1*	*Running*	*0*
cert-manager-cainjector-784f978fb7-rf9zr	*1/1*	*Running*	*0*
cert-manager-webhook-767f9dbc85-4w78q	*1/1*	*Running*	*0*
cert-manager-webhook-767f9dbc85-hkxkt	*1/1*	*Running*	*0*
cert-manager-webhook-767f9dbc85-hnnc2	*1/1*	*Running*	*0*

Install cmctl command-line tool

This process needs to be completed on all control-plane nodes

1. SSH into all control-plane node
2. Download binary
   - 2.1. Run **sudo apt update && sudo apt install -y golang**
   - 2.2. Run **mkdir -p $HOME/projects/k8s/framework/cert-manager/v1.17.0/cmctl && cd $HOME/projects/k8s/framework/cert-manager/v1.17.0/cmctl**
   - 2.3. Download and extract cmctl
     - 2.3.1. Run **OS=$(uname -s | tr A-Z a-z); ARCH=$(uname -m | sed 's/x86_64/amd64/' | sed 's/aarch64/arm64/'); curl -fsSL -o cmctl https://github.com/cert-manager/cmctl/releases/latest/download/cmctl_${OS}_${ARCH}**
     - 2.3.2. Run **chmod +x cmctl**
     - 2.3.3. Run **sudo mv cmctl /usr/local/bin/kubectl-cert_manager**
3. Activate cmctl shell completion for plugin mode
   - 3.1. Run **kubectl cert-manager completion kubectl > kubectl_complete-cert_manager**
   - 3.2. Run **sudo install kubectl_complete-cert_manager /usr/local/bin**
4. Verify
   - 4.1. Run **kubectl cert-manager** <TAB>

Install IngressController-NGINX

NOTE:

Kubernetes ingress development has been frozen and no new features will be developed. Any new features for exposing services will be developed for Gateway API. Seriously consider migrating any ingress services to a Gateway API.

https://kubernetes.io/docs/concepts/services-networking/ingress/

https://gateway-api.sigs.k8s.io/implementations/

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/nginx-ingresscontroller/helm**
3. Install Ingress-NGINX via Helm
   - 3.1. Run **helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx**
   - 3.2. Run **helm repo update**

3.3. Run **helm upgrade -i ingress-nginx-controller ingress-nginx/ingress-nginx --version 4.12.0 -n ingress-nginx --create-namespace -f helm-ingress-nginx-override-values.yaml**

4. Wait for pods to be operational
   4.1. Run **kubectl wait --namespace ingress-nginx --for=condition=ready pod --selector=app.kubernetes.io/component=controller --timeout=120s**
5. Verify all pods
   5.1. Run **kubectl get pods -n ingress-nginx**
      5.1.1. You should receive similar output:

```
NAME                                                    READY   STATUS
ingress-nginx-controller-controller-f6bf54f45-vvbht      1/1    Running
ingress-nginx-controller-controller-f6bf54f45-xfqkz      1/1    Running
ingress-nginx-controller-defaultbackend-75db55c867-422k4 1/1    Running
```

Install Rook-Ceph Storage

Prerequisites

1. Installation location
   1.1. Rook will only be installed **ONCE** on any control plane node. Once installed, the operator will proceed to install/configure the remainder nodes
2. Linux packages
   2.1. lvm2
      2.1.1. ** Needs to exist on ALL nodes in the cluster **
   2.2. git (client)
3. Raw storage
   3.1. Raw devices (no partitions or formatted filesystem)
   3.2. Block storage presented to storage nodes
      3.2.1. We are taking the assumption you want to present two different types of backend storage (SSD and mechanical). If you are only presenting one tier of storage, then you can modify override-values.yaml to accommodate.

         SCSI **sdc** = SSD disk type => pool ‘rbd-ssd’
         SCSI **sdd** = 7.2K disk type => pool ‘rbd-hdd’
         SCSI **sde** = 7.2K disk type => pool ‘cephfs-hdd’
         SCSI **sdf** = SSD disk type => pool ‘mgr-metadata’ (used only for .mgr)

   3.3. Ensure the hard disk are represented to the correct location in the kernel
      3.3.1. SSH into each worker node
      3.3.2. Run **lsblk**
         3.3.2.1. You should receive similar output:

```
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0       7:0    0 91.8M  1 loop /snap/lxd/23991
loop1       7:1    0 55.6M  1 loop /snap/core18/2566
loop3       7:3    0 67.8M  1 loop /snap/lxd/22753
loop4       7:4    0   48M  1 loop /snap/snapd/17336
loop5       7:5    0 63.2M  1 loop /snap/core20/1634
loop6       7:6    0 49.7M  1 loop /snap/snapd/17576
loop7       7:7    0 55.6M  1 loop /snap/core18/2632
loop8       7:8    0 63.2M  1 loop /snap/core20/1695
sda         8:0    0  150G  0 disk
├─sda1      8:1    0    1M  0 part
├─sda2      8:2    0    1G  0 part /boot
```

```
└─sda3                 8:3    0  149G  0 part
  └─ubuntu--vg-ubuntu--lv 253:0    0  149G  0 lvm  /
sdc                    8:32   0  100G  0 disk
sdd                    8:48   0  150G  0 disk
sde                    8:64  0  75GB
sdf                    8:80 0  10GB
sr0                   11:0    1 1024M  0 rom
```

**** Rook will capture *ALL* storage devices that meet the above stipulations and create a pool ****

4. ***Production cluster requires a minimum of 4 storage nodes***
   4.1. This is because Rook requires multiple nodes for secure replication of data
5. Cert-Manger needs to be installed for the Rook admission controller
   5.1. Use the **Install and Configure Cert-Manager** section

Install CSI-Addons Controller

The csi-addons version is tied very closely too the version of Rook-Ceph being deployed. Ensure the versions are compatible with each other.

https://rook.io/docs/rook/latest-release/Storage-Configuration/Ceph-CSI/ceph-csi-drivers/#csi-addons-controller

1. SSH into a control-plane node
2. Run **mkdir -p $HOME/projects/k8s/rook/csi-addons/v0.11.0 && cd ~/projects/k8s/rook/csi-addons/v0.11.0/**
3. Clone the repository
   3.1. Run **git clone --single-branch --branch v0.11.0 https://github.com/csi-addons/kubernetes-csi-addons.git**
4. Create CSI-Addon controller objects
   4.1. Run **cd $HOME/projects/k8s/rook/csi-addons/v0.11.0/kubernetes-csi-addons/deploy/controller**
   4.2. Run **kubectl apply -f crds.yaml -f setup-controller.yaml -f csi-addons-config.yaml -f rbac.yaml**
   Need to specify the order of deployment with this version as applying at the directory level will attempt to create objects before parent objects exist
5. Verify CSI-Addons components
   5.1. Run **kubectl wait --namespace csi-addons-system --for=condition=ready pod --selector=app.kubernetes.io/name=csi-addons --timeout=120s**

Install Kubernetes-CSI External-Snapshotter

1. SSH into a control-plane node
2. Run **mkdir -p $HOME/projects/k8s/kubernetes-csi-external-snapshotter/v8.2.0/ && cd $HOME/projects/k8s/kubernetes-csi-external-snapshotter/v8.2.0/**
3. Run **git clone --single-branch --branch v8.2.0 https://github.com/kubernetes-csi/external-snapshotter.git**
4. Install Snapshot CRDs
   4.1. Run **kubectl kustomize external-snapshotter/client/config/crd | kubectl create -f -**
5. Install Common Snapshot Controller
   5.1. Run **kubectl -n kube-system kustomize external-snapshotter/deploy/kubernetes/snapshot-controller | kubectl create -f -**
6. Verification
   6.1. Run **kubectl get pods -n kube-system -l app.kubernetes.io/name=snapshot-controller**

6.1.1. You should receive similar output:

NAME	READY	STATUS	RESTARTS	AGE
snapshot-controller	*1/1*	*Running*	*0*	*14m*
snapshot-controller	*1/1*	*Running*	*0*	*14m*

Installation (Helm)

1. SSH into a control-plane node
2. Source files
   2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/rook-ceph/helm**
3. Add label to any node that will run Rook MGR, OSD, or MON. This will usually be all (or a subset) of your storage nodes.
   3.1. Run **kubectl label node** <k8 node name> **role-storage=rook-node**
   3.2. Run **kubectl label node** <k8 node name> **role-rook-osd=rook-osd-node**
4. Add taint to your storage nodes. This will prevent everyday workload (pods) from running on these nodes and possibly overwhelm them
   4.1. Run **kubectl taint nodes** <k8 node name> **role-storage=rook-node:NoSchedule**
5. Add Helm repository
   5.1. Run **helm repo add rook-release https://charts.rook.io/release**
   5.2. Run **helm repo update**

Installation Rook Operator

1. SSH into a control-plane node
2. Source files
   2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/rook-ceph/helm**
3. Install the Rook operator
   3.1. Run **helm upgrade -i rook-ceph rook-release/rook-ceph -n rook-ceph --create-namespace --version 1.16.1 -f helm-rook-ceph-operator-override-values.yaml**
4. Verify operator
   4.1. Run **kubectl wait --namespace rook-ceph --for=condition=ready pod --selector=app=rook-ceph-operator --timeout=120s**

Installation Rook Cluster

1. SSH into a control-plane node
2. Source files
   2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/rook-ceph/helm**
3. Install and configure the Rook-Ceph cluster
   3.1. Run **helm upgrade -i rook-ceph-cluster rook-release/rook-ceph-cluster -n rook-ceph --version 1.16.1 -f helm-rook-ceph-cluster-override-values.yaml**
4. Verify Rook cluster
   4.1. This process can take upto ~5min to complete
   4.2. Run **kubectl get -n rook-ceph cephblockpools.ceph.rook.io**
      4.2.1. You should receive similar output:

NAME	PHASE
replicapool-hdd	***Ready***
replicapool-ssd	***Ready***
replicapool-mgrmeta	*Ready*

4.3. Run **kubectl get -n rook-ceph cephfilesystems.ceph.rook.io**

4.3.1. You should receive similar output:

NAME	ACTIVEMDS	PHASE
cephfspool-hdd	*1*	*Ready*

4.4. Run **kubectl get storageclasses.storage.k8s.io**

4.4.1. You should receive similar output:

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEEXPANSION
rookceph-cfs-hdd	*rook-ceph.cephfs.csi.ceph.com*	*Delete*	*Immediate*	*true*
rookceph-rbd-hdd (default)	*rook-ceph.rbd.csi.ceph.com*	*Delete*	*Immediate*	*true*
rookceph-rbd-ssd	*rook-ceph.rbd.csi.ceph.com*	*Delete*	*Immediate*	*true*

4.5. Run **kubectl get volumesnapshotclasses.snapshot.storage.k8s.io**

4.5.1. You should receive similar output:

NAME	DRIVER	DELETIONPOLICY
csi-rbdplugin-snapclass	***rook-ceph.rbd.csi.ceph.com***	*Delete*
csi-cephfsplugin-snapclass	***rook-ceph.cephfs.csi.ceph.com***	*Delete*

4.6. Run **kubectl get pods -n rook-ceph -o custom-columns='POD:.metadata.name,CONTAINER:.spec.containers[*].name' | grep "csi-addons"**

4.6.1. You should receive similar output:

POD	CONTAINER
csi-cephfsplugin-provisioner	*csi-attacher,csi-snapshotter,csi-resizer,csi-provisioner,csi-cephfsplugin,**csi-addons***
csi-cephfsplugin-provisioner	*csi-attacher,csi-snapshotter,csi-resizer,csi-provisioner,csi-cephfsplugin,**csi-addons***
csi-rbdplugin-provisioner	*csi-provisioner,csi-resizer,csi-attacher,csi-snapshotter,**csi-addons**,csi-rbdplugin*
csi-rbdplugin-provisioner	*csi-provisioner,csi-resizer,csi-attacher,csi-snapshotter,**csi-addons**,csi-rbdplugin*
csi-rbdplugin-	*driver-registrar,csi-rbdplugin,**csi-addons***
csi-rbdplugin-	*driver-registrar,csi-rbdplugin,**csi-addons***
csi-rbdplugin-	*driver-registrar,csi-rbdplugin,**csi-addons***
csi-rbdplugin-	*driver-registrar,csi-rbdplugin,**csi-addons***

Rook Toolbox – Enable Orchestrator

1. Run **kubectl exec -it --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- /bin/sh**
   1.1. This will open a shell terminal into the container
   1.2. Run **ceph status**
      1.2.1. You should receive similar output:

```
cluster:
  id:     ad07d635-36e3-486e-9164-b945faeb6324
  health: HEALTH_OK
```

services:
mon: 3 daemons, quorum a,b,c (age 16m)
mgr: a(active, since 15m), standbys: b
osd: 12 osds: 12 up (since 15m), 12 in (since 15m)

data:
pools: 1 pools, 1 pgs
objects: 2 objects, 449 KiB
usage: 1.0 GiB used, 2.1 TiB / 2.1 TiB avail
pgs: 1 active+clean

1.3. Run **ceph mgr module enable rook**
1.4. Run **ceph telemetry on --license sharing-1-0**
1.5. Run **ceph telemetry enable channel perf**
1.6. Run **ceph orch set backend rook**
1.7. Validate Orchestrator status
   1.7.1. Run **/usr/bin/ceph orch status**
      1.7.1.1. You should receive similar output:

Backend: rook
Available: Yes

   1.7.2. Run **ceph mgr module ls**
      1.7.2.1. You should receive similar output:

MODULE
balancer on (always on)
crash on (always on)
devicehealth on (always on)
orchestrator on (always on)
pg_autoscaler on (always on)

1.8. Run **exit**

Install Rook-Ceph kubectl plugin

These steps need to be completed on all control-plane nodes

1. SSH into a control plane node
2. Install KREW package manager
   2.1. Run

```
(
 set -x; cd "$(mktemp -d)" &&
 OS="$(uname | tr '[:upper:]' '[:lower:]')" &&
 ARCH="$(uname -m | sed -e 's/x86_64/amd64/' -e 's/\(arm\)\(64\)\?.*/\1\2/' -e
's/aarch64$/arm64/')" &&
 KREW="krew-${OS}_${ARCH}" &&
 curl -fsSLO "https://github.com/kubernetes-
sigs/krew/releases/latest/download/${KREW}.tar.gz" &&
 tar zxvf "${KREW}.tar.gz" &&
 ./"${KREW}" install krew
)
```

2.2. Modify ~/.bashrc (or .zshrc) to add the path

2.2.1. Run **sed -i '$a export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"' ~/.bashrc**

2.2.2. Run **exec bash**

2.3. Verify Krew

2.3.1. Run **kubectl krew update**

2.3.1.1. You should receive similar output:

Updated the local copy of plugin index.

3. Install Rook-Ceph kubectl plugin

3.1. Run **kubectl krew install rook-ceph**

3.2. Verify

3.2.1. Run **kubectl rook-ceph ceph status**

Move default .mgr pool to custom CRUSH rule

This is very important for this setup

Because we have configured multiple custom CRUSH rules via setting DeviceSets, the '.mgr' pool will use the default CRUSH rule and root. This will cause overlapping roots for the cluster and break auto-balancing and PG placement.

1. SSH into a control-plane node
2. Run **kubectl exec --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- ceph osd pool set .mgr crush_rule .mgr**
3. Verify

3.1. Run **kubectl exec --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- ceph osd pool autoscale-status**

3.1.1. You should receive similar output:

POOL	*SIZE*	*RATE*	*RAW CAPACITY*	*BIAS*	*PG_NUM*	*AUTOSCALE*	*BULK*
replicapool-ssd	*19*	*3.0*	*600.0G 0.0000*	*1.0*	*32*	***on***	*False*
replicapool-hdd	*242.9M*	*3.0*	*1800G 0.0004*	*1.0*	*32*	***on***	*False*
cephfspool-hdd-metadata	*71185*	*3.0*	*225.0G 0.0000*	*4.0*	*16*	***on***	*False*

3.1.1.1. If you receive no output, more than likely, there are still overlapping roots and the autobalance is skipping pools

Verify Rook Storage Consumable

Deploy a test pod that will consume and mount 2Gi of each storageclass.

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/rook-ceph/tools**
3. Deploy the appropriate test pod

3.1. There are 2 different test pod [**all** | **rbd**]

3.1.1. rbd = **for when you only have RBD pools deployed**

3.1.1.1. Run **kubectl apply -f rook-test-rbd-storageclass-consumable.yaml**

3.1.2. all = **for when both RBD and CephFS pools deployed**

3.1.2.1. Run **kubectl apply -f rook-test-all-storageclass-consumable.yaml**

4. Verify
    4.1. Verify pod
        4.1.1. Run **kubectl get po -n default rook-storageclass-consumer**
            4.1.1.1. Should be in the running and ready status
    4.2. Verify PVC/PV
        4.2.1. Run **kubectl get pv**
            4.2.1.1. You should receive similar output:

NAME	*CAPACITY*	*ACCESS*	*STATUS*	*CLAIM*	*STORAGECLASS*
pvc-xxxx	*2Gi*	*RWO*	*Bound*	*default/rook-storageclass-consumer-pv-claim-rbd-hdd*	*rook-ceph-rbd-hdd*
pvc-xxxx	*2Gi*	*RWO*	*Bound*	*default/rook-storageclass-consumer-pv-claim-cfs-hdd*	*rook-ceph-cfs-hdd*
pvc-xxxx	*2Gi*	*RWO*	*Bound*	*default/rook-storageclass-consumer-pv-claim-rbd-ssd*	*rook-ceph-rbd-ssd*

1. If verification fails with PV not bound and in a 'pending' status
    1.1. This might be caused by the Rook provisioners being stuck
        1.1.1. Run **kubectl get po -n rook-ceph**
        1.1.2. Locate the pods with the name prefix of: (should be 4 of them)
            1.1.2.1. csi-cephfsplugin-provisioner-
            1.1.2.2. csi-rbdplugin-provisioner-
        1.1.3. One at a time
            1.1.3.1. Run **kubectl delete -n rook-ceph pod** <pod name>
    1.2. Re-run the Rook test pod deployment
2. Remove the Rook test pod and storage
    2.1. rbd = for when you only have RBD pools deployed
        2.1.1. Run **kubectl delete -f rook-test-rbd-storageclass-consumable.yaml**
    2.2. all = for when both RBD and CephFS pools deployed
        2.2.1. Run **kubectl delete -f rook-test-all-storageclass-consumable.yaml**

Rook-Ceph UI Dashboard

Install Rook-Ceph Dashboard

If you installed Rook via the operator, then the dashboard should already be installed.

1. SSH into a control plane node
2. Verify the Rook dashboard service
    2.1. Run **kubectl get svc -n rook-ceph rook-ceph-mgr-dashboard**
        2.1.1. You should receive similar output:

NAME	*TYPE*	*CLUSTER-IP*	*EXTERNAL-IP*	*PORT(S)*	*AGE*
rook-ceph-mgr-dashboard	*ClusterIP*	*172.16.215.226*	*<none>*	*8443/TCP*	*15d*

1. Verify the Rook dashboard ingress
    1.1. Run **kubectl get -n rook-ceph ingress**
        1.1.1. You should receive similar output:

NAME	*CLASS*	*HOSTS*	*PORTS*
rook-ceph-dashboard	*nginxx*	*k8-clst100-rook-ceph-ceph.abiwot-lab.com*	*80*

Expose Rook-Ceph UI

Rook Dashboard UI Login and Verification

1. SSH into a control-plane node
2. Acquire UI login token
    2.1. Run **kubectl -n rook-ceph get secret rook-ceph-dashboard-password -o jsonpath="{['data']['password']}" | base64 --decode && echo**
        2.1.1. Copy output and **keep secure**
3. Navigate to the UI via a browser
    3.1. Navigate to **https://cdak8clst100-rook-ceph-ceph.**<domain>**/**
        3.1.1. Default username = 'admin'
        3.1.2. Password is the output captured above
    3.2. The main dashboard should show the overall health as 'healthy'

Install Kubernetes Dashboard

Helm installation of K8s-dashboard

1. SSH into a control-plane node
2. Run **helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/**
3. Run **helm repo update**
4. Run **helm upgrade -i kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard -n kubernetes-dashboard --create-namespace --version 7.10.3 --set metrics-server.enabled=true --set metrics-server.args[0]=--kubelet-insecure-tls**

Create User & Role for Kubernetes Dashboard

Create User – admin-user

This user will be an Administrator level user within dashboard and cluster. **This account is to be used only by Kubernetes Admins.** Users should access the dashboard with their account via Dex.

Create Service Account – admin-user

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/k8-dashboard/**
3. Run **kubectl apply -f svcacct_k8_dashboard_admin-user.yaml**
4. Validate the service account was created
    4.1. Run **kubectl get serviceaccounts -n kubernetes-dashboard**
        4.1.1. You should receive similar output:

NAME	SECRETS	AGE
admin-user	0	7m23s
default	0	48m
kubernetes-dashboard	**0**	**48m**

Create Bearer Token for Kubernetes Dashboard – admin-user

1. SSH into a control-plane node

2. Run **kubectl -n kubernetes-dashboard create token admin-user**
   2.1. Copy output and **keep secure**

Expose Kubernetes Dashboard UI

Configure Ingress-controller to expose Kubernetes-dashboard UI

1. SSH into a control plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/k8-dashboard/**
3. Run **kubectl apply -f ingress-k8dashboard-ui.yaml**
4. Validate the Ingress
   4.1. Run **kubectl get ingress -n kubernetes-dashboard**
      4.1.1. You should receive similar output:

NAME	CLASS	HOSTS	PORTS
k8-dashboard-ui-nodeport-expose	***nginx***	***cdak8clst100-k8dashboard.abiwot-lab.com***	***80***

Kubernetes Metric Server

This feature is now a part of the default install of the Kubernetes Dashboard platform.

Once the dashboard is installed, you can now also run the metrics from the cli.

1. SSH into a control-plane node
2. Run **kubectl top nodes**
   2.1. Output will be of all K8s nodes with CPU and memory
3. Run **kubectl top pods -A**

Output will be of all pods with CPU and memory

Install and Configure Prometheus

Install Prometheus

1. SSH into any control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/**
3. Add the Helm chart
   3.1. Run **helm repo add prometheus-community https://prometheus-community.github.io/helm-charts**
   3.2. Run **helm repo update**
4. Install Prometheus
   4.1. Run **helm upgrade -i kube-prometheus prometheus-community/kube-prometheus-stack --values helm/helm-kube-prome-stack-override-values.yaml -n monitoring --create-namespace --version 68.4.3**
5. Verify Prometheus pods are operational
   5.1. Run **kubectl --namespace monitoring get pods -l "release=kube-prometheus"**
      5.1.1. You should receive similar output:

NAME	*READY*	*STATUS*	*RESTARTS*
alertmanager-kube-prometheus-kube-prome-alertmanager-0	*2/2*	*Running*	*0*
kube-prometheus-grafana-57f9fbc585-skwhz	*3/3*	*Running*	*0*
kube-prometheus-kube-prome-operator-5d55d4548b-bh62w	*1/1*	*Running*	*0*
kube-prometheus-kube-state-metrics-6fb56b5c8c-dxwdx	*1/1*	*Running*	*0*

kube-prometheus-prometheus-node-exporter-52b6g 1/1 Running 0
kube-prometheus-prometheus-node-exporter-9p7cl 1/1 Running 0
kube-prometheus-prometheus-node-exporter-c5jxp 1/1 Running 0
kube-prometheus-prometheus-node-exporter-cvdmj 1/1 Running 0
kube-prometheus-prometheus-node-exporter-dgm4q 1/1 Running 0
kube-prometheus-prometheus-node-exporter-xjr7b 1/1 Running 0
kube-prometheus-prometheus-node-exporter-zrlnq 1/1 Running 0
prometheus-kube-prometheus-kube-prome-prometheus-0 2/2 Running 0

Expose Prometheus via Ingress Controller

Expose Prometheus Components

Expose AlertManager UI

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/**
3. Create the ingress
    3.1. Run **kubectl apply -f ingress/ingress-alertmanager-ui.yaml**
4. Verify AlertManager UI exposure
    4.1. Launch a browser window with network access to the k8s cluster or external load-balancer
        4.1.1. Navigate to **https://cdak8clst100-monitoring-alertmanager.abiwot-lab.com/**

Expose Prometheus UI

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/**
3. Create the ingress
    3.1. Run **kubectl apply -f ingress/ingress-prometheus-ui.yaml**
4. Verify Prometheus UI exposure
    4.1. Launch a browser window with network access to the k8s cluster or external load-balancer
        4.1.1. Navigate to **https://cdak8clst100-monitoring-prometheus.abiwot-lab.com/**
5. Verify Prometheus Targets
    5.1. Navigate within Prometheus UI -> Status -> Target Health
        5.1.1. View the list and ensure all are reporting as ‘up’
            5.1.1.1. Some of the K8s core components will show as ‘down’. This is expected at this point and will addressed in the section “Monitor Kubernetes Core Components”

Expose Grafana UI

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/**
3. Create the ingress
    3.1. Run **kubectl apply -f ingress/ingress-grafana-ui.yaml**
4. Verify Prometheus UI exposure
    4.1. Launch a browser window with network access to the k8s cluster or external load-balancer
        4.1.1. Navigate to **https://cdak8clst100-monitoring-grafana.abiwot-lab.com/**
5. Verify Grafana graphs

5.1. SSH into a control-plane node
5.2. Acquire Grafana default login secret
5.2.1. Grafana username
5.2.1.1. Run **echo $(kubectl get secret --namespace monitoring kube-prometheus-grafana -o jsonpath='{.data.admin-user}' | base64 -d)**
5.2.2. Grafana password
5.2.2.1. Run **echo $(kubectl get secret --namespace monitoring kube-prometheus-grafana -o jsonpath='{.data.admin-password}' | base64 -d)**
5.3. On your browser window, login to the Grafana UI with the credentials acquired above
5.3.1. Navigate to General -> Kubernetes -> Compute Resources -> Cluster

Monitor Kubernetes Core Components

Prometheus cannot scrape Kubernetes core components by default, since these pods are bound to localhost.

Note of Caution:

Ensure you understand the possible security implications of exposing the "/metrics" on the control-plane external IP. Since this K8s deployment has disabled anonymous API access, access to the "/metrics" still requires a token but, that alone is not the ultimate security guard.

1. Kube-controller-manager
   1.1. **Duplicate this step on all control-plane nodes**
   1.2. Need to change the bind-address on the kube-controller-manager
   1.2.1. Run **sudo sed -i 's/- --bind-address=127.0.0.1/- --bind-address=0.0.0.0/' /etc/kubernetes/manifests/kube-controller-manager.yaml**
2. Kube-scheduler
   2.1. **Duplicate this step on all control-plane nodes**
   2.2. Need to change the bind-address on the kube-scheduler
   2.2.1. Run **sudo sed -i 's/- --bind-address=127.0.0.1/- --bind-address=0.0.0.0/' /etc/kubernetes/manifests/kube-scheduler.yaml**
3. kube-proxy
   3.1. **This step only needs to be executed once for the cluster**
   3.2. Need to expose the kube-proxy metrics flag
   3.2.1. Run **kubectl get configmap -n kube-system kube-proxy -o yaml > $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/kube-proxy-cm.yaml**
   3.2.2. Run **sed -i 's\metricsBindAddress:.*\metricsBindAddress: "0.0.0.0:10249"\' $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/kube-proxy-cm.yaml**
   3.2.3. Run **kubectl apply -f $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/kube-proxy-cm.yaml**
   3.3. Reset kube-proxy pods since configmap changes are applied during pod creation
   3.3.1. Run **kubectl -n kube-system delete po -l k8s-app=kube-proxy**
4. etcd
   4.1. **This step only needs to be executed once for the cluster**
   4.2. Metrics from etcd requires a daemonset to facilitate the scraping of the metrics.
   4.2.1. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/kube-prometheus-stack/kube-rbac-proxy**
   4.3. Create service account for etcd daemonsets and deploy daemonsets
   4.3.1. Run **kubectl apply -f kube-rbac-proxy-svcacct.yaml -f kube-rbac-proxy-daemonset-etcd.yaml**

5. Verify Prometheus targets
   5.1. Navigate back to the Prometheus UI – Status > Target Health
      5.1.1. All targets should no be reporting UP with metrics

Additional Prometheus Monitoring and Grafana Dashboards

The management intention of this Prometheus stack is that each component (Calico, Rook, K8sPacket, etc…) will control their needs into monitoring.

Prometheus monitoring:

The general idea is any Prometheus targets you want to add to monitoring:

1. Create a service to gather metrics from a pod/daemonset
2. Create a service-monitor to gather the metrics via the service
3. The service-monitor requires the following to ensure it is automatically added
   3.1. Label
      3.1.1. **release: kube-prometheus**
   3.2. Namespace
      3.2.1. **monitoring**
      3.2.2. If you use a different namespace, then you need to add a service-account, with the appropriate permissions and/or network security policy to allow Prometheus access

Grafana dashboards:

The general idea is any Grafana dashboards you want to add to monitoring:

1. Create a configmap with the JSON format of the dashboard as the data
2. For the configmap to automatically add the dashboard, it requires:
   2.1. Label
      2.1.1. **grafana_dashboard: “1”**
   2.2. Namespace
      2.2.1. **monitoring**

Configure Calico for Prometheus monitoring

Service and Service-monitor Configuration

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/tigera-calico/prometheus**
3. Configure Felix metrics
   3.1. Deploy a service to target all the daemonsets of calico-node
      3.1.1. Run **kubectl apply -f felix-metrics-svc.yaml**
   3.2. Deploy a service-monitor to capture the endpoints of the Felix service
      3.2.1. Run **kubectl apply -f svcmon-felix-metrics-monitor.yaml**
4. Configure Typha metrics
   4.1. There will already be a service calico-system/calico-typha-metrics. This is deployed via the Helm override values within tigera-operator
   4.2. Deploy a service-monitor to capture the endpoints of the Typha service
      4.2.1. Run **kubectl apply -f svcmon-typha-metrics-monitor.yaml**

5. Configure Calico kube-controllers metrics
   5.1. There will already be a service calico-system/calico-kube-controllers-metrics. This is deployed via the default Helm values.
   5.2. Deploy a service-monitor to capture the endpoints of the Calico kube-controllers
      5.2.1. Run **kubectl apply -f svcmon-calico-kube-controllers-metrics-monitor.yaml**
6. Verify Prometheus targets
   6.1. From the Prometheus UI – Status > Target Health
   6.2. Locate the following new metrics
      6.2.1. **serviceMonitor/monitoring/calico-kube-controllers-metrics-monitor**
      6.2.2. **serviceMonitor/monitoring/felix-metrics-monitor**
      6.2.3. serviceMonitor/monitoring/typha-metrics-monitor
   6.3. It could take 1-2 minutes for the new targets to show in Prometheus

Grafana Chart Configuration

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/tigera-calico/prometheus**
   2.1. Run **kubectl apply -f dashboards/**
3. Validate dashboard
   3.1. Navigate to the Grafana UI
      3.1.1. Navigate to Home > Dashboards > Felix Dashboard (Calico)

Configure Rook-Ceph for Prometheus Monitoring

These instructions will allow a central Prometheus stack to monitor and alert on the Rook-Ceph deployment.

Enable Rook-Ceph Operator and Cluster Level Monitoring

Rook-Ceph operator and cluster level monitoring settings require Prometheus to pre-exist before enabling, these steps must be phased.

Pay special attention to the Helm command flag **'--reuse-values'**. This will essentially merge your existing values with any new ones.

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/rook-ceph/prometheus/**
3. Enable monitoring on operator level (CSI metrics)
   3.1. Run **helm upgrade -i rook-ceph rook-release/rook-ceph -n rook-ceph --version 1.16.1 --reuse-values -f helm-rook-ceph-operator-monitoring.yaml**
4. Verify CSI targets in Prometheus
   4.1. Run **kubectl get servicemonitors.monitoring.coreos.com -n monitoring csi-metrics**
   4.2. Navigate to Prometheus UI > Status > Target Health
      4.2.1. You should see targets for **serviceMonitor/monitoring/csi-metrics**
         4.2.1.1. It can take 1-2 minutes for the targets to populate
5. Enable monitoring on the cluster level
   5.1. Run **helm upgrade -i rook-ceph-cluster rook-release/rook-ceph-cluster -n rook-ceph --version 1.16.1 --reuse-values -f helm-rook-ceph-cluster-monitoring.yaml**
6. Label Ceph service-monitors

6.1. Currently there is no option to add labels to the service-monitors (via Helm) created for the ceph-cluster. So we will create a service-account and cronjob to add the **'release: kube-prometheus'**.

6.2. Run **kubectl apply -f rbac-servicemonitor-patch.yaml -f cronjob-patch-rook-sm.yaml**

7. Verify Ceph targets in Prometheus
   7.1. Navigate to Prometheus UI > Status > Target Health
      7.1.1. You should see targets for
         7.1.1.1. **serviceMonitor/rook-ceph/rook-ceph-exporter**
         7.1.1.2. **serviceMonitor/rook-ceph/rook-ceph-mgr**
8. Verify Prometheus rules for Ceph
   8.1. Navigate to Prometheus UI > Alerts
      8.1.1. You should see new rules for (rados, pools, pgs, etc...)

Ceph UI Observability

1. If you want to receive Ceph alerts within the Ceph UI, you have to point the resource to the Prometheus stack. In our deployment, we are going to use the AlertManager and Prometheus external address (same URL you use to get to the UI)
   1.1. Run **kubectl exec --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- ceph dashboard set-alertmanager-api-host 'https://cdak8clst100-monitoring-alertmanager.abiwot-lab.com'**
   1.2. Run **kubectl exec --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- ceph dashboard set-alertmanager-api-ssl-verify False**
   1.3. Run **kubectl exec --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- ceph dashboard set-prometheus-api-host 'https://cdak8clst100-monitoring-prometheus.abiwot-lab.com'**
   1.4. Run **kubectl exec --namespace=rook-ceph -it $(kubectl get pod --namespace=rook-ceph -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- ceph dashboard set-prometheus-api-ssl-verify False**
2. Verify Alerts
   2.1. Navigate to the Ceph UI > Observability > Alerts
      2.1.1. It can take a few min for this page to populate. You might also have to log out of the UI and back in

Configure Ceph Grafana dashboards

There are some basic dashboards to be installed (Ceph-cluster, Ceph-osd, & Ceph-pools) from the Grafana labs repository. These are maintained by the Rook community. These install instructions will use the sidecar method by creating configmaps within K8s for Grafana to automatically pickup.

See README for details how to create the configmap files

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/rook-ceph/prometheus/**
3. Install configmaps
   3.1. Run **kubectl apply -f dashboards/**
4. Verification
   4.1. Navigate to your Grafana UI and login
   4.2. Navigate to Dashboards > Browse
      4.2.1. You should see 3 new graphs
         4.2.1.1. Ceph – Cluster
         4.2.1.2. Ceph – OSD (Single)
         4.2.1.3. Ceph – Pools

4.3. Click on each one to verify data is populated in each graph. Some graph sub-charts might take 1-2 min to populate

Configure CSI Addon-ons Metrics

Since we are using the provided service-monitor manifest from the csi-addons project, we need to modify a few aspects, on the fly, before being applied. The following command basically creates a new service-monitor manifest with additional label for Prometheus and deployed in the csi-addons-system namespace

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/rook/csi-addons/v0.11.0/kubernetes-csi-addons/config/prometheus**
3. Modify the monitor.yaml file
   3.1. Run **kubectl apply -f monitor.yaml --dry-run=client -o yaml | kubectl label -f - --dry-run=client -o yaml --local release=kube-prometheus | yq eval '.metadata.namespace = "csi-addons-system"' - > custom-monitor.yaml**
4. Deploy the service-monitor
   4.1. Run **kubectl apply -f custom-monitor.yaml**
5. Verify service-monitor deployment
   5.1. Run **kubectl get servicemonitors.monitoring.coreos.com -n csi-addons-system controller-manager-metrics-monitor**
6. Verify Prometheus Targets
   6.1. Navigate to Prometheus UI > Status > Target Health
      6.1.1. You should see **serviceMonitor/csi-addons-system/controller-manager-metrics-monitor**

Configure NGINX-ingress Metrics

These instructions will allow a central Prometheus stack to monitor and alert on the NGINX-ingress deployment.

Prometheus Configurations

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/nginx-ingresscontroller/prometheus**
3. Merge the NGINX-ingress Helm deployment for monitoring
   3.1. Run **helm upgrade -i ingress-nginx-controller ingress-nginx/ingress-nginx --version 4.12.0 -n ingress-nginx --reuse-values -f helm-ingress-nginx-monitoring.yaml**
4. Validate monitoring components
   4.1. Run **kubectl get svc -n ingress-nginx -o wide ingress-nginx-controller-controller-metrics**
   4.2. Run **kubectl get servicemonitors.monitoring.coreos.com -n ingress-nginx ingress-nginx-controller-controller**
5. Validate Prometheus objects
   5.1. Targets
      5.1.1. Navigate to Prometheus UI > Status > Target Health
         5.1.1.1. You should see **serviceMonitor/ingress-nginx/ingress-nginx-controller-controller**
   5.2. Alert Rules
      5.2.1. Navigate to Prometheus UI > Alerts
         5.2.1.1. You should see **ingress-nginx**

Grafana Configurations

1. SSH into a control-plane node
2. Run **cd $HOME/projects/k8s/abiwot-kubeadm/framework/nginx-ingresscontroller/prometheus**
3. Run **kubectl apply -f dashboards/**
4. Validate Grafana dashboard
   4.1. Navigate to Grafana UI > Home > Dashboards
      4.1.1. You should see **Kubernetes Nginx Ingress Prometheus NextGen**

Configure Cert-Manager Metrics

These instructions will allow a central Prometheus stack monitor and alert on Cert-Manger deployment.

Prometheus Configurations

1. SSH into a control-plane node
2. Run **cd $HOME/ projects/k8s/abiwot-kubeadm/framework/cert-manager/prometheus**
3. Run **helm upgrade -i cert-manager jetstack/cert-manager -n cert-manager --create-namespace --version v1.17.0 -f helm-cert-manager-monitoring-override-values.yaml --reuse-values**
4. Verify Prometheus targets
    4.1. Navigate to Prometheus UI > Status > Target Health
        4.1.1. You should see **serviceMonitor/cert-manager/cert-manager**

Grafana Configurations

1. SSH into a control-plane node
2. Run **cd $HOME/ projects/k8s/abiwot-kubeadm/framework/cert-manager/prometheus**
3. Run **kubectl apply -f dashboards/**
4. Validate Grafana dashboard
    4.1. Navigate to Grafana UI > Home > Dashboards
        4.1.1. You should see **Cert-manager-Kubernetes**

(left intentionally blank)

Appendix A:

Files:

(left intentionally blank)