|  |  |
| --- | --- |
| Kubernetes Infra Setup Guide  Prepared By:  KPIT Technologies Ltd. |  |
| VERSION : Draft  DATE : 23-09-2024  Prepared For:  <PFS>  <Pune>, <India> |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | |
|  |  | | |
|  |  |  |  |
|  |
|  |
|  |  | | |

Contents

[1. Introduction 3](#_Toc177946504)

[1.1 architecture & Components: 3](#_Toc177946505)

[1.2 CONTROL PLANE (Master node) COMPONENETS: 3](#_Toc177946506)

[1.3 worker node COMPONENETS: 4](#_Toc177946507)

[2. Cluster setup 5](#_Toc177946508)

[2.1 Prerequisties: 5](#_Toc177946509)

[2.2 steps follow on Both master & worker servers: 5](#_Toc177946510)

# Introduction

This Guide aims to provide a comprehensive walkthrough for setting up a robust and scalable Kubernetes infrastructure within our organization. It is the leading open-source platform for container orchestration, empowers organizations to automate the deployment scaling and management of containerized applications across diverse environments.

1.1 **architecture & Components:**

A diagram of a diagram

Description automatically generated

REFERENCE: <https://kubernetes.io/docs/concepts/architecture/>

1.2 **CONTROL PLANE (Master node) COMPONENETS:**

In Kubernetes, a master node is the **control plane** responsible for **managing the cluster**. It coordinates and schedules tasks, maintains cluster state, and monitors node health. It includes components like API server, scheduler, and controller manager, ensuring overall cluster functionality and orchestration of containerized applications.

**1. Kube API server:** it handles administrative tasks on the master node. Users send REST commands in YAML/JSON to the API server, which processes and executes them. The Kube API server acts as the front end of the Kubernetes control plane.

**2.** **etcd**: it is a useful tool for storing key-value data in a distributed approach. It’s designed for storing data about Kubernetes clusters, such as information about pods, their state, and namespaces. etcd is only accessible from the API server to maintain security.

Kubernetes uses etcd to manage its key-value API through gRPC. All objects are stored under the /registry directory key in key-value format.

The API-server of Kubernetes uses the etcd’s watch feature to monitor any modifications to an object’s state. As the sole Stateful set component in the control plane, etcd is an excellent database for Kubernetes.

**3.** **Kube-scheduler:** When deploying a pod in a Kubernetes cluster, the kube-scheduler identifies the best worker node that satisfies the pod requirements, such as CPU, memory, and affinity. Upon identification, it schedules the pod on the right node. This process is made possible because of etcd’s role in storing vital information needed by Kubernetes to run smoothly. The necessary information is stored in the etcd file cabinet whenever a request is made to Kubernetes.

**4. Kube Controller Manager:** its task is to retrieve the desired state from the API Server. If the desired state does not match the current state of the object, corrective steps are taken by the control loop to align the current state with the desired state.

1.3 **worker node COMPONENETS:**

**1. Kubelet:** The kubelet is an essential component that runs on every node in the Kubernetes cluster. It acts as an agent responsible for registering worker nodes with the API server and working with the podSpec primarily from the API server.

The kubelet creates, modifies, and deletes containers for the pod. Additionally, it handles liveliness, readiness, and startup probes. It also mounts volumes by reading pod configuration and creating respective directories reporting Node pod status via calls to the API server**.**

The kubelet can accept podSpec from various sources, including a file, HTTP endpoint, and HTTP server. It also uses the node’s container runtime to pull images, run containers, etc.

The Kubelet starts the API-server, scheduler, and controller manager as static pods while bootstrapping the control plane. The kubelet is crucial in managing the containers and ensuring the pod is in the desired state.

**2. Kube proxy:** kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

**3. Container runtime:** A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

# Cluster setup

2.1 **Prerequisties:**

* Three Linux machines (Ubuntu 22.04 LTS)
  + 1 Master Node (Kubernetes API, Scheduler, controller Manager, etc.)
  + 1 Worker Node (Where the application workloads will run)
  + 1 Load balancer Node for HAProxy (For distributing traffic across your Kubernetes nodes)
* Root/sudo access on all nodes.
* Stable network connectivity between nodes.
* Internet access for installing packages.

2.2 **steps for setup:**

* **Update the system**:

``sudo apt update && sudo apt upgrade -y``

* **Disable Swap:**

``sudo swapoff -a``

``sudo sed -I ‘/ swap / s/^/#/’ /etc/fstab``

* **Load kernel Modules:**

``sudo modprobe overlay``

``sudo modprobe br\_netfilter``

* **Update sysctl settings:**

``sudo tee /etc/sysctl.d/kubernetes.conf << EOF

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

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

net.ipv4.ip\_forward = 1

EOF``

``sudo sysctl –system``

* **Install Docker (As Kubernetes requires container runtime):**

``sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates``

``sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/docker.gpg``

``sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" ``

``sudo apt update``

``sudo apt install -y containerd.io``

* **Start & enable Docker:**

``sudo systemctl enable docker``

``sudo systemctl start docker``

* **Updating the apt package:**

``sudo apt-get update``

* **Install the required packages that enables secure connections for Kubernetes apt repository by below command**

``sudo apt-get install -y apt-transport-https ca-certificate curl gpg``

* **Add the Kubernetes APT Repository:**

# Add the GPG key

``curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg ``

# Add the Kubernetes APT repository

`` echo “deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /” | sudo tee /etc/apt/sources.list.d/kubernetes.list ``

**Note:** If the directory “/etc/apt/keyrings” does not exist, it should be created before adding the Kubernetes GPG keys. This directory used to store GPG keys securely

`` sudo mkdir -p -m 755 /etc/apt/keyrings``

* **Installing kubeadm, kubelet and kubectl:**

`` sudo apt-get update``

`` sudo apt-get install -y kubelet kubeadm kubectl``

`` sudo apt-mark hold kubelet kubeadm kubectl``

* **Initialize the Cluster :**

``sudo systemctl enable –now kubelet``(optional)

# Run below command on Master Node:

``sudo kubeadm init --pod-network-cidr=10.10.0.0/16``

# Set up kubectl for the current user:

`` mkdir -p $HOME/.kube``

``sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config``

``sudo chown $(id -u):$(id -g) $HOME/.kube/config``

# Status of the cluster:

`` kubectl cluster-info``

``kubectl get nodes``

* **Join the Worker node to the cluster:**

# On the Worker node run below command:

``sudo kubeadm join <master-ip>:6443 –token <token> --discovery-token-ca-cert-hash sha256:<hash>``

* **Install Calico Networking on the Master Node :**

**Note:** Download the Calico manifest (calico.yaml) for Kubernetes networking by below curl command which download the yaml file in your current directory for configuration or deployment in your kubernetes cluster

``curl -o https://docs.projectcalico.org/manifest/calico.yaml``  
  
you must find the section ``CALICO\_IPV4POOL\_CIDR`` in your yaml file. It is an important configuration in Calico networking that specifies the range of IPs used for pod networking. Adjust it based on your network architecture to ensure smooth IP allocation and avoid conflicts.

---

# The default IPv4 pool to create on startup if none exists. Pod IPs will be #chosen from this range. Changing this value after installation will have

#no effect. This should fall within `--cluster-cidr`.

# - name: CALICO\_IPV4POOL\_CIDR

# value: "192.168.0.0/16"

# Disable file logging so `kubectl logs` works.

- name: CALICO\_DISABLE\_FILE\_LOGGING

value: 'true'

We correct the IP range to match the CIDR of the pod network in the command sudo kubeadm init. In my example, 10.10.0.0/16the file after editing has the following form

---

# The default IPv4 pool to create on startup if none exists. Pod IPs will be

# chosen from this range. Changing this value after installation will have

# no effect. This should fall within `--cluster-cidr`.

- name: CALICO\_IPV4POOL\_CIDR

value: '10.10.0.0/16'

# Disable file logging so `kubectl logs` works.

- name: CALICO\_DISABLE\_FILE\_LOGGING

value: 'true'

**Install Calico on the Master node:**

``kubectl apply -f calico.yaml``

We will check to see if calico has been deployed successfully by checking the pods on the namespace kube-system

-master:~$ kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

calico-kube-controllers-57b57c56f-ptddp 1/1 Running 0 2m44s

calico-node-5fqml 1/1 Running 0 2m44s

calico-node-llfjq 1/1 Running 0 2m44s

calico-node-vw78h 1/1 Running 0 2m44s

coredns-787d4945fb-n7s9t 1/1 Running 0 62m

If the status is Running, it means the deployment has been successful. Now if you check the status of the nodes, the status will be Ready

master:~$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

k8s-master.nvtienanh.local Ready control-plane 62m v1.26.1

k8s-worker1.nvtienanh.local Ready <none> 5m53s v1.26.1

k8s-worker2.nvtienanh.local Ready <none> 5m10s v1.26.1