# kubernetes-documentation

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Kubernetes provides you with:

1. Service discovery and load balancing Kubernetes can expose a container using the DNS name or using their own IP address.

If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.

2. Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.

3. Automated rollouts and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

4. Automatic bin packing You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.

5. Self-healing Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

# Description

The aim of the project was to create the kubernetes cluster with three nodes and deploy the kubernetes dashboard to monitor the resources and the nodes.

# Prerequisites

. A fresh Installation of Rocky Linux 9

. Sudo user with admin rights

. Minimum of 2 GB RAM, 2 vCPUs and 20 GB Disk Space

. A reliable Internet Connection

# Lab Setup

We have used three Virtual machines with following specification

. K8s-masternode 192.168.1.150

. K8s-node2 k8s-masternodek8s-node1k8s-node2 192.168.1.100

K8s-node1 192.168.1.200

```

### Step 2: Disable Swap Space on Each Node

For kubelet to work smoothly, we must disable swap space on all the nodes. Run beneath command

```bash

sudo swapoff -a

```

The above comand it disable swap when the computer is on when it boot you have to disable swap again

```bash

sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab

```

The above command it disable swap command everytime the computer boot

### Step 3: Adjust SELinux and Firewall Rules for Kubernetes

Set SELinux mode as permissive on all the nodes using following commands

```bash

sudo setenforce 0

sudo sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=permissive/g' /etc/sysconfig/selinux

```

On the master node, allow following ports in the firewall

```bash

sudo firewall-cmd --permanent --add-port={6443,2379,2380,10250,10251,10252,10257,10259,179}/tcp

sudo firewall-cmd --permanent --add-port=4789/udp

sudo firewall-cmd --reload

```

On the Worker Nodes, allow beneath ports in the firewall

```bash

sudo firewall-cmd --permanent --add-port={179,10250,30000-32767}/tcp

sudo firewall-cmd --permanent --add-port=4789/udp

sudo firewall-cmd --reload

```

### Step 4: Add Kernel Modules and Parameters

For kuberetes cluster, we must add the overlay and br\_netfilter kernel modules on all the nodes.

Create a file and add following content to it

```bash

sudo tee /etc/modules-load.d/containerd.conf <<EOF

overlay

br\_netfilter

EOF

```

In order to load above modules, run

```bash

sudo modprobe overlay

sudo modprobe br\_netfilter

```

Next, add the following kernel parameters, create a file and with following content

```bash

sudo vi /etc/sysctl.d/k8s.conf

```

copy inside the following content inside the file

```bash

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

net.ipv4.ip\_forward = 1

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

```

Save & close the file.

Now add these parameters by running below command

```bash

sudo sysctl --system

```

### Step 5: Install Conatinerd Runtime

Kubernetes requires a container runtime, and one of the most popular choices is containerd.

But It is not available in the default package repositories of Rocky Linux so add the following docker repo on all the nodes.

```bash

sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

```

Now, run following dnf command to install containerd on all the nodes

```bash

sudo dnf install containerd.io -y

```

Configure containerd so that it will use systemdcgroup, execute the following commands on each node

```bash

containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1

sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml

```

Restart and enable containerd service using beneath commands

```bash

sudo systemctl restart containerd

sudo systemctl enable containerd

```

Verify conatinerd service status, run

```bash

sudo systemctl status containerd

```

### Step 6: Install Kubernetes tools

Kubernetes tools like Kubeadm, kubectl and kubelet are not available in the default package repositories of Rocky Linux 9.

So, to install these tools, add the following repository on all the nodes.

```bash

# Add the Kubernetes repository to all nodes

cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/

enabled=1

gpgcheck=1

gpgkey=https://pkgs.k8s.io/core:/stable:/v1.31/rpm/repodata/repomd.xml.key

EOF

```

Note: At time of writing this post, Kubernetes 1.31 version was available, thatNoteReadReadys test our Kubernetes installation the next step.

### Step 9: Test Kubernetes Cluster Installation

To test Kubernetes cluster installation deploy a simple pod to ensure that the cluster can schedule and run workloads.

Create a sample YAML file for an nginx pod

```bash

cat <<EOF > nginx-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

EOF

```

Deploy the pod

```bash

kubectl apply -f nginx-pod.yaml

```

Verify that the pod is running

```bash

kubectl get pods

```

# Kubernetes Dashboard installation

In order to get the kubernetes dashboard find the kuberdashboard repo and clone it

Clone K8s-Dashboard repo to deploy it with automation:

```bash

git clone https://github.com/irsols-devops/kubernetes-dashboard.git

```

## Details of K8s dashboard manual steps with NodePort

1. Create the deployment :

```bash

kubectl apply -f kubernetes-dashboard-deployment.yml

```

1.1. Create the Admin User and Role Binding:

First, ensure that the admin-user service account and cluster role binding are correctly applied. You can create the following YAML file (admin-user.yaml) to ensure it's set up properly:

yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kubernetes-dashboard

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin-user

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin-user

namespace: kubernetes-dashboard

Apply this YAML:

bash

kubectl apply -f admin-user.yaml

1.2. Generate a Token for the Admin User:

After applying the service account and cluster role binding, create a new token for the admin-user if it does not already exist.

Create a YAML file (admin-user-token.yaml) like this:

yaml

apiVersion: v1

kind: Secret

metadata:

name: admin-user-token

namespace: kubernetes-dashboard

annotations:

kubernetes.io/service-account.name: admin-user

type: kubernetes.io/service-account-token

Apply the token creation:

bash

kubectl apply -f admin-user-token.yaml

1.3. Retrieve the Admin Token:

Now, retrieve the token for the admin-user:

bash

kubectl -n kubernetes-dashboard describe secret admin-user-token

This command should return a secret with a token under the Data section. Copy this token and use it to log in to the Kubernetes Dashboard.

2. Create service account admin role :

```bash

kubectl apply -f service-account-admin.yml

```

3. Perform cluster role binding :

```bash

kubectl apply -f rbac-clusterRoleBinding-admin.yml

```

4. Read through the YAML files and apply to your cluster to install & configure dashboard properly:

```bash

kubectl apply -f ./kubernetes-dashboard/

```

5. Ensure K8s deployment and service are up and running :

```bash

kubectl get deployments -n kubernetes-dashboard

```

6. If this is a multi-node cluster then figure out which node the dashboard pods are running on :

```bash

kubectl get pods --namespace=kubernetes-dashboard -o wide

```

Double check that there are no issues in the services and its not in CrashLoopBack state

7. Get the NodePort that dashboard is exposed on using following :

```bash

kubectl get services --namespace=kubernetes-dashboard -o wide

```

8. Get service token and store in a local file using :

```bash

kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep admin-user | awk '{print $1}')

```

9. Open up a browser that allows self-signed certificate using webpages ( e.g. Firefox ) and use the node IP:Nodeport combination to access the dashboard .