**Lab Guide for Docker Setup and Node.js App Deployment**

**1. Provision a Virtual Server**

* **Cloud Provider**: AWS, GCP, or Azure
* **Instance Configuration**:
  + OS: Ubuntu
  + Storage: 10GB
  + vCPU: 1
  + RAM: 1GB

**2. Connect to the Server**

* Open your terminal (or use SSH client).
* Connect to the server using SSH:

bash

Copy code

ssh ubuntu@<server-ip-address>

* Switch to root user:

bash

Copy code

sudo -i

**3. Install Docker on Ubuntu**

* Follow the official Docker installation guide for Ubuntu: [Docker Installation Guide](https://docs.docker.com/engine/install/ubuntu/)
* **Commands**:

bash

Copy code

sudo apt-get update

sudo apt-get install ca-certificates curl gnupg

sudo install -m 0755 -d /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

* Verify Docker installation:

bash

Copy code

docker -v

docker ps

**4. Clone the GitHub Repository**

* Clone your Node.js project repository:

bash

Copy code

git clone https://github.com/ramannkhanna2/nodejsapp\_jenkins-docker-kubernetes.git

cd nodejsapp\_jenkins-docker-kubernetes

**5. Understand Docker Run Flags**

* The docker run command can be used with various flags. Some commonly used ones:
  + **-i**: Keep STDIN open even if not attached.
  + **-t**: Allocate a pseudo-TTY (makes it interactive).
  + **-d**: Run container in detached mode (in the background).

**6. Inspect Dockerfile**

* The Dockerfile describes the instructions for building a Docker image for your Node.js application.

Dockerfile

Copy code

FROM node:latest

WORKDIR /usr/src/app

COPY package.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "index.js"]

* **Explanation**:
  + FROM node:latest: Use the latest Node.js base image.
  + WORKDIR /usr/src/app: Set the working directory.
  + COPY package.json ./: Copy the package.json to the working directory.
  + RUN npm install: Install Node.js dependencies.
  + COPY . .: Copy all files from your local directory to the container.
  + EXPOSE 3000: Expose port 3000 for the application.
  + CMD ["node", "index.js"]: Command to start the application.

**7. Build the Docker Image**

* Run the following command to build the image from the Dockerfile in the current directory. Tag the image with customimage.

bash

Copy code

docker build -t customimage .

* Verify the image is built:

bash

Copy code

docker images

**8. Run the Docker Container**

* Start a container from the customimage Docker image:

bash

Copy code

docker run -d -p 3000:3000 --name mynodeapp customimage

* **Explanation**:
  + -d: Run in detached mode.
  + -p 3000:3000: Map port 3000 of the container to port 3000 of the host.

**9. Verify Application Deployment**

* List running containers:

bash

Copy code

docker ps

* Retrieve the container IP address:

bash

Copy code

docker inspect mynodeapp | grep IPAddress

* **Access the Application**:
  + In your browser or with curl, access the application via the server’s IP and port:

bash

Copy code

curl http://<server-ip>:3000

**10. Additional Docker Operations**

* **Interactive Access**: Open a shell inside the container.

bash

Copy code

docker exec -it mynodeapp /bin/bash

* **Check Container Stats**: Monitor container resource usage.

bash

Copy code

docker stats

* **Stop and Remove Containers**:
  + Stop a container:

bash

Copy code

docker stop mynodeapp

* + Remove a container:

bash

Copy code

docker rm mynodeapp

**11. Explore the Application Routes**

* If the application has additional routes, test them similarly with curl or a browser.

bash

Copy code

curl http://<server-ip>:3000/ready

**12. Clean Up**

* Stop all containers:

bash

Copy code

docker stop $(docker ps -q)

* Remove all containers and images if necessary:

bash

Copy code

docker rm $(docker ps -a -q)

docker rmi $(docker images -q)

This lab guide provides a complete workflow to set up, build, and deploy a Dockerized Node.js application on a cloud-hosted Ubuntu server, along with essential Docker operations and cleanup steps.

4o

**Kubernetes Cluster Setup with Kubeadm: Detailed Lab Guide**

This guide outlines the setup of a Kubernetes cluster using kubeadm, with a master node and one or more worker nodes. Each node will run on Ubuntu and use Docker as the container runtime, configured with cri-dockerd. The Calico network plugin will be installed for pod networking.

**Prerequisites**

1. Provision Ubuntu servers (master and worker nodes).
   * **Recommended Configuration:** 10 GB storage, 1 vCPU, and 1 GB RAM for each.
   * Install docker.io and kubeadm, kubectl, and kubelet.
2. **Network Requirements:**
   * Ensure all nodes can communicate over the internal network.
   * Port 6443 must be open on the master node (Kubernetes API server).
   * Other ports as per the [Kubernetes networking documentation](https://kubernetes.io/docs/reference/networking/ports-and-protocols/).

**1. Kubernetes Setup Script (Run on Each Node)**

To set up Docker and Kubernetes components on each node, create and run the following script:

bash

Copy code

# Create the setup script

vi script.sh

# Paste the following into script.sh

#!/bin/bash

# Update package lists and install Docker

sudo apt update -y

sudo apt install docker.io -y

# Set up Kubernetes repository and install kubelet, kubeadm, and kubectl

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

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

# Enable net.bridge.bridge-nf-call-iptables for networking

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

# Install cri-dockerd

wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.14/cri-dockerd-0.3.14.amd64.tgz

tar -xvf cri-dockerd-0.3.14.amd64.tgz

cd cri-dockerd

sudo install -o root -g root -m 0755 cri-dockerd /usr/local/bin/cri-dockerd

# Set up cri-dockerd systemd service

cd ..

wget https://github.com/Mirantis/cri-dockerd/archive/refs/tags/v0.3.14.tar.gz

tar -xvf v0.3.14.tar.gz

cd cri-dockerd-0.3.14/

sudo cp packaging/systemd/\* /etc/systemd/system

sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service

# Enable and start cri-docker service

sudo systemctl daemon-reload

sudo systemctl enable --now cri-docker.socket

sudo systemctl enable cri-docker

sudo systemctl start cri-docker

sudo systemctl status cri-docker

* Save and exit the editor (:wq).
* Make the script executable and run it:

bash

Copy code

chmod +x script.sh

./script.sh

This installs Docker, kubelet, kubeadm, kubectl, and configures cri-dockerd as the container runtime.

**2. Master Node: Configuration and Initialization**

On the master node, create a kubeadm configuration file config.yaml to initialize the cluster:

bash

Copy code

vi config.yaml

Paste the following configuration into config.yaml, replacing advertiseAddress with the private IP of the master node:

yaml

Copy code

apiVersion: kubeadm.k8s.io/v1beta3

kind: InitConfiguration

localAPIEndpoint:

advertiseAddress: <MASTER\_NODE\_PRIVATE\_IP>

bindPort: 6443

nodeRegistration:

criSocket: unix:///var/run/cri-dockerd.sock

---

apiVersion: kubeadm.k8s.io/v1beta3

kind: ClusterConfiguration

networking:

podSubnet: 192.168.0.0/16

Save and exit the file (:wq).

**Initialize the Master Node:**

bash

Copy code

kubeadm init --config=config.yaml | tee cluster\_initialized.txt

This will generate the necessary token and CA hash required for worker nodes to join the cluster.

**3. Configure kubectl on Master Node**

To use kubectl commands on the master node, configure the kubeconfig file:

bash

Copy code

mkdir -p $HOME/.kube

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

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

Verify that the master node is set up correctly:

bash

Copy code

kubectl get nodes

**4. Networking: Install Calico Plugin**

Calico provides networking for pod communication. Install it on the master node:

bash

Copy code

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml

**5. Join Worker Nodes**

On each worker node, use the kubeadm join command generated in cluster\_initialized.txt to join the cluster. The command will look similar to this:

bash

Copy code

kubeadm join <MASTER\_NODE\_PRIVATE\_IP>:6443 --token <TOKEN> \

--discovery-token-ca-cert-hash sha256:<HASH> \

--cri-socket unix:///var/run/cri-dockerd.sock

Replace <MASTER\_NODE\_PRIVATE\_IP>, <TOKEN>, and <HASH> with the values from your initialization output.

**6. Verify Cluster and Nodes**

On the master node, check the status of nodes and pods:

bash

Copy code

kubectl get nodes -A

kubectl get pods -A -o wide

**Basic Kubernetes Commands for Node and Pod Management**

* **List all nodes:**

bash

Copy code

kubectl get nodes

* **List all pods across namespaces:**

bash

Copy code

kubectl get pods -A

* **Describe a specific node:**

bash

Copy code

kubectl describe node <node-name>

* **Inspect a specific pod:**

bash

Copy code

kubectl describe pod <pod-name> -n <namespace>

**Additional Notes**

* **Managing the Cluster**: Familiarize yourself with core management tasks like scaling, deploying applications, and managing resources.
* **Networking and Ports**: Review the [Kubernetes ports and protocols documentation](https://kubernetes.io/docs/reference/networking/ports-and-protocols/) for networking details, especially for managing firewall rules.

This setup prepares a Kubernetes cluster with Docker as the container runtime, Calico as the network plugin, and kubeadm for simplified setup and node management.