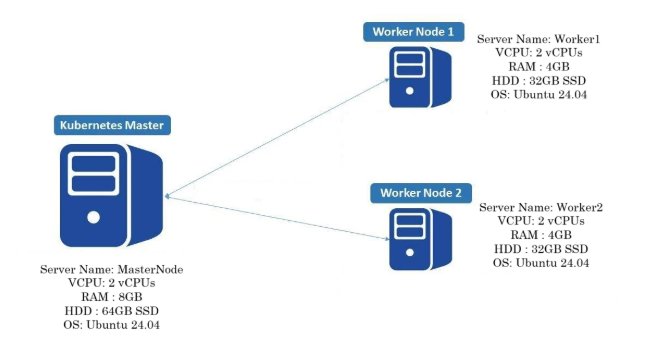
<https://www.linkedin.com/pulse/setting-up-kubernetes-cluster-azure-using-ubuntu-2404-velumani-bkric/?trackingId=JKhnW1yoQeedGdrmNr1V%2Bg%3D%3D>



Setting Up a Kubernetes Cluster on Azure Using Ubuntu 24.04 VMs: A Step-by-Step Guide

Kubernetes (K8s) is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. In this guide, we will walk you through the process of setting up Kubernetes on Azure using Ubuntu 24.04 virtual machines (VMs). By the end of this tutorial, you will have a fully operational Kubernetes cluster capable of managing your applications and services.

Kubernetes Manual Setup vs Managed Kubernetes Services

While managed services like EKS, GKE, and AKS simplify Kubernetes management and abstract away many complexities, setting up Kubernetes manually provides more control, flexibility, and a deeper understanding of how Kubernetes operates.

When you set up Kubernetes manually:

* You gain full control over the cluster’s configuration and components.
* You can fine-tune networking and security policies to meet specific application requirements.
* You have flexibility in managing workloads, scaling, and integrating custom services.
* It’s an excellent learning opportunity, giving you hands-on experience with Kubernetes components like kubeadm, kubelet, and kubectl.

In contrast, using managed Kubernetes services like EKS, GKE, or AKS is easier for production-grade environments but may limit control and customization.

Lab Setup Overview

We will configure a Kubernetes cluster with one master node and two worker nodes, all running Ubuntu 24.04:

* **Master Node:** MasterNode (Control Plane)
* **Worker Node 1:** Worker1
* **Worker Node 2:** Worker2

**Step 1: Create a Resource Group**

Start by creating a resource group in Azure to organize your resources (e.g., VMs, Network Security Groups).

>> az group create --name K8-Lab --location southeast-asia

**Step 2: Set Up Virtual Machines in Azure**

**Create Virtual Machines**: In the Azure portal, create three virtual machines: one for the master node and two for the worker nodes.

**Master Node**: 2 vCPUs | 8 GB RAM | 64 GB SSD

**Worker Nodes**: 2 vCPUs | 4 GB RAM | 32 GB SSD

**Step 3: Create Network Security Group (NSG)**

To ensure proper communication, configure the following inbound and outbound rules for the NSG:

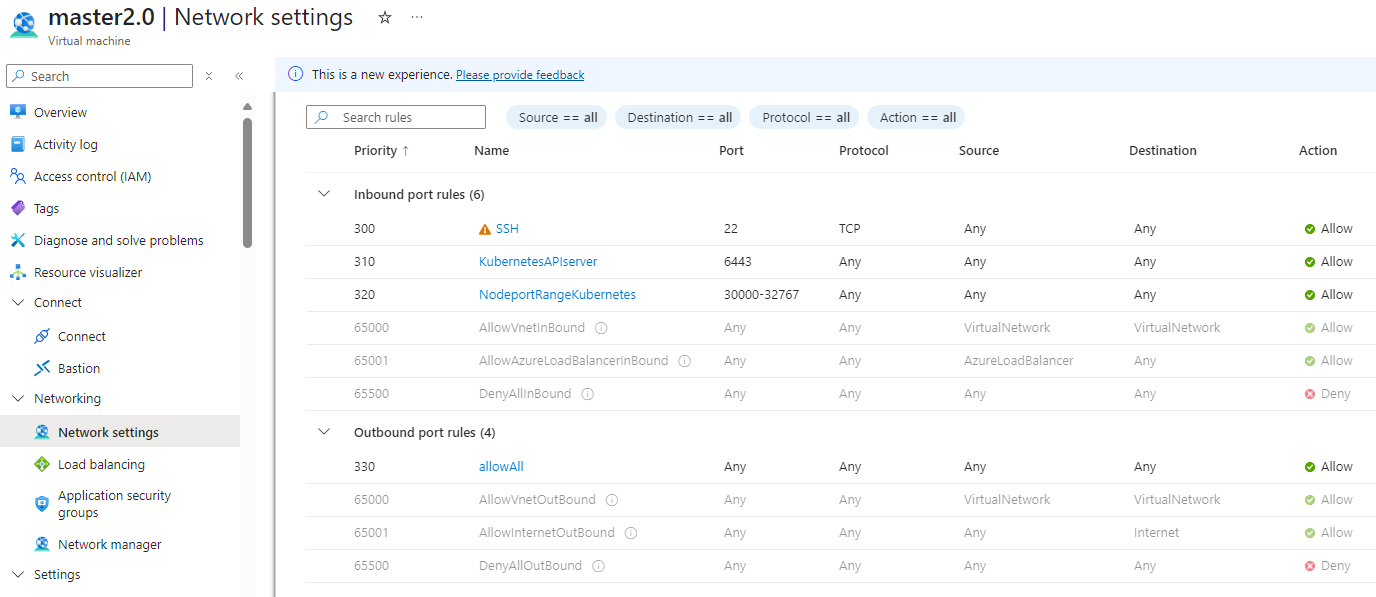
**Inbound NSG Rules:**

* Port 6443: Kubernetes API server
* Port range 30000-32767: NodePort range for Kubernetes services
* Port 22: SSH for remote login

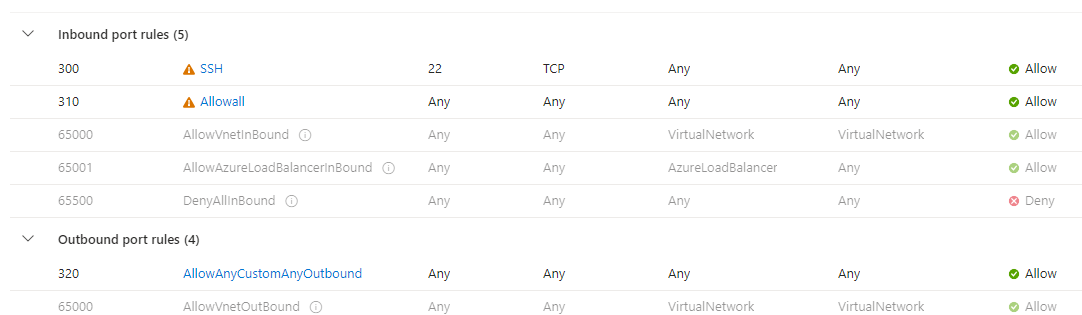
**Outbound NSG Rules:**

* Allow all outbound traffic.

Master-



Worker-



**Step 4: Prepare the Nodes for Cluster Setup**

**Set Hostnames:** Log in to each VM and set the appropriate hostname. For example, on the master node:

>> sudo hostnamectl set-hostname MasterNode // In MasterNode

>> exec bash // In MasterNode

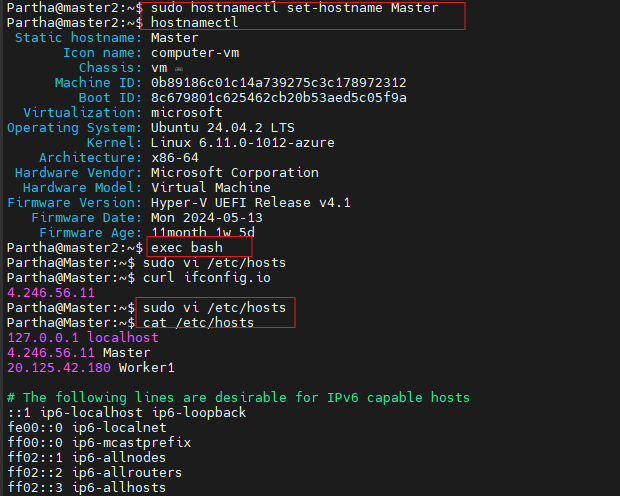
>> sudo hostnamectl set-hostname WorkerNode1 // In Worker1

>> exec bash // In Worker1

>> sudo hostnamectl set-hostname WorkerNode2 // In Worker2

>> exec bash // In Worker2

Note- Use Public IP



**Edit** /etc/hosts: Ensure the /etc/hosts file on each node contains the correct entries for all nodes:

>> sudo nano /etc/hosts

Add the following entries, replacing <MasterNode-ip>, <Worker1-ip>, and <Worker2-ip> with the actual IP addresses:

<MasterNode-ip> MasterNode

<Worker1-ip> Worker1

<Worker2-ip> Worker2

Note- same for master and worker

After making the changes, press CTRL+X to exit, then press Y to confirm saving, and hit Enter to save the file. Repeat this process on all nodes in your Kubernetes cluster.

**Disable Swap:** Kubernetes requires swap to be disabled on all nodes:

>> sudo swapoff -a

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

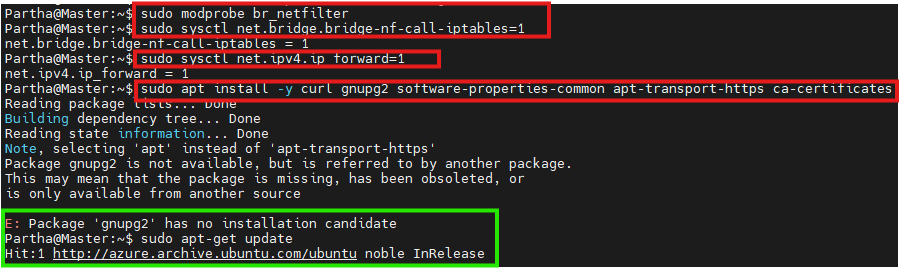
****

**Load Kernel Modules and Set Parameters:** Run the following commands to load the necessary kernel modules and set parameters on all nodes:

>> sudo modprobe br\_netfilter

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

>> sudo sysctl net.ipv4.ip\_forward=1



Note- if error try to update first “sudo apt-get update”.

**Step 5: Install Containerd Runtime**

Kubernetes uses containerd to manage containers. Install it using the following commands:

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

# Enable Docker repository

>> sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmour -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"

# Install containerd

>> sudo apt update

>> sudo apt install -y containerd.io

# Configure containerd with systemd

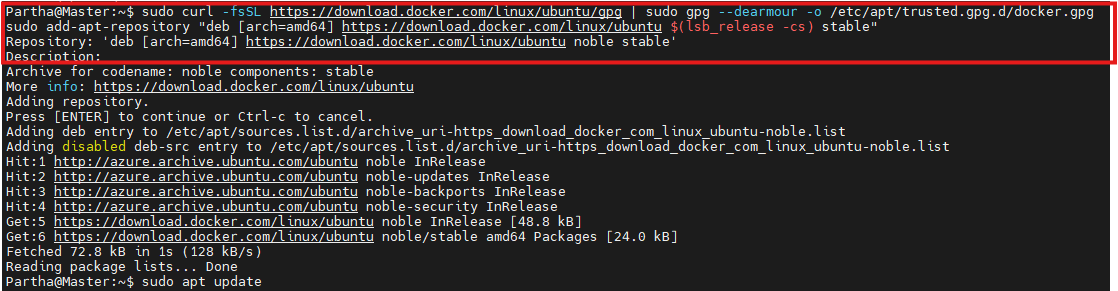
>> containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1

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

# Restart and enable containerd

>> sudo systemctl restart containerd

>> sudo systemctl enable containerd

****

**Step 6: Install Kubernetes Components**

Add the Kubernetes repository and install the necessary components (kubectl, kubeadm, and kubelet):

>> curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/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:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

>> sudo apt update

>> sudo apt install -y kubelet kubeadm kubectl

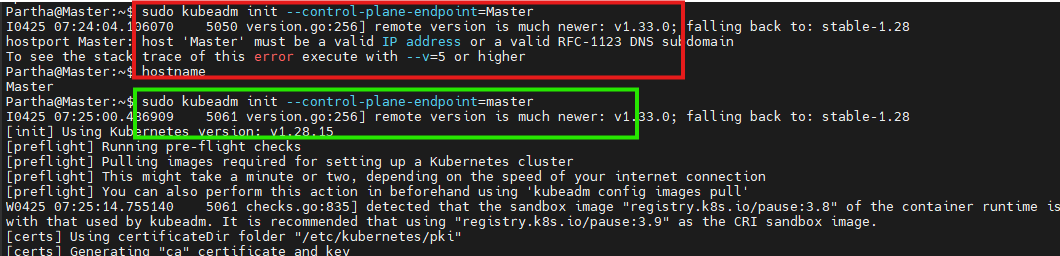
>> sudo apt-mark hold kubelet kubeadm kubectl

**Step 7: Initialize the Kubernetes Cluster**

**Initialize the Master Node:** On the MasterNode, run the following command to initialize the control plane:

>> sudo kubeadm init --control-plane-endpoint=<your-master-node-name>

Ex- sudo kubeadm init --control-plane-endpoint=master



If any error. Try node name small letters.

**Configure** kubectl: After initialization, configure kubectl to interact with the cluster:

>> mkdir -p $HOME/.kube

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

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

**Verify Cluster and Node Status:**

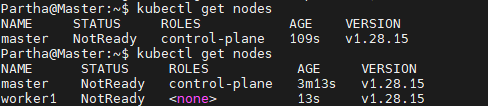
>> kubectl cluster-info

>> kubectl get nodes

**Step 8: Join Worker Nodes to the Cluster**

On each worker node (Worker1, Worker2), run the kubeadm join command obtained from the MasterNode’s kubeadm init output:

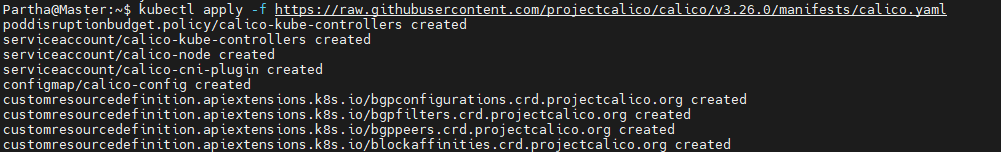
>> sudo kubeadm join MasterNode:6443 --token <your-token> --discovery-token-ca-cert-hash sha256:<your-hash>



**Step 9: Install a Network Plugin (Calico)**

Kubernetes requires a CNI (Container Network Interface) plugin for pod-to-pod communication. We’ll use Calico for this purpose:

>> kubectl apply -f <https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml>



**Step 10: Test the Cluster**

Deploy a test Nginx application to verify the cluster is working correctly:

>> kubectl create deployment nginx-app --image=nginx --replicas=2

>> kubectl expose deployment nginx-app --type=NodePort --port=80

Check the service and access it:

>> kubectl get svc nginx-app kubectl describe svc nginx-app

Access the service using:

curl http://<worker-node-ip>:<node-port>

**Troubleshooting Common Issues**

Setting up Kubernetes manually on Azure comes with its challenges. Below are common issues and their solutions:

**1. Port In Use**

**Problem:** Ports required by Kubernetes services (e.g., 10250, 2380) are already in use.

**Solution:**

Stop the conflicting services:

sudo systemctl stop kubelet

sudo systemctl stop etcd

Clear old Kubernetes data:

sudo rm -rf /var/lib/etcd/\*

Run kubeadm init with --ignore-preflight-errors=all to bypass errors:

sudo kubeadm init --control-plane-endpoint=MasterNode --ignore-preflight-errors=all

**2. Existing Kubernetes Manifests Interfering with New Setup**

**Problem:** Old manifests or data from a previous setup interfere with the new cluster.

**Solution:** Clear old manifests and data:

sudo rm -f /etc/kubernetes/manifests/\*

**3. kubeadm init Fails Due to Swap**

**Problem:** Swap is enabled, which Kubernetes doesn’t support.

**Solution:** Disable swap:

sudo swapoff -a

Permanently disable swap by editing /etc/fstab.

Re-run kubeadm init:

sudo kubeadm init --control-plane-endpoint=MasterNode --ignore-preflight-errors=all

**4. Worker Nodes Failing to Join**

**Problem:** Worker nodes fail to join due to incorrect join command.

**Solution:** Retrieve the correct join token and certificate hash from the MasterNode:

sudo kubeadm token list openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl dgst -sha256

On each worker node, run the kubeadm join command:

sudo kubeadm join MasterNode:6443 --token <your-token> --discovery-token-ca-cert-hash sha256:<your-hash>

**5. Node IP Address Conflicts**

**Problem:** Node IPs conflict due to incorrect /etc/hosts entries or network misconfigurations.

**Solution:** Ensure each node has the correct entry in /etc/hosts. Recheck network connectivity using ping or curl.

**6. Incorrect Kubernetes Component Installation**

**Problem:** Kubernetes components (kubeadm, kubelet, kubectl) fail to install or are incompatible.

**Solution:** Ensure you are using compatible versions of kubeadm, kubelet, and kubectl. You can install specific versions with:

sudo apt install -y kubeadm=<version> kubelet=<version> kubectl=<version>

**7. Insufficient Resources**

**Problem:** VMs don’t meet the resource requirements for Kubernetes components

**Solution:** Ensure each VM has at least 2 vCPUs and 4 GB RAM. If necessary, resize the VMs or add more resources to the VM.

**8. Firewall Blocks Kubernetes Communication**

**Problem:** Firewall rules prevent necessary Kubernetes ports from being open.

**Solution:** Verify that the firewall allows communication on ports used by Kubernetes services, including 6443 (API server) and 2379-2380 (etcd).

**9. CNI Plugin Issues**

**Problem:** The network plugin (e.g., Calico) fails to deploy.

**Solution:** Ensure the required CNI plugin is compatible with your Kubernetes version. Re-apply the network plugin manifest:

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml

**10. Node Disconnection**

**Problem:** A node is unexpectedly disconnected from the cluster.

**Solution:** Check the node’s system logs (journalctl -u kubelet) for errors and try restarting the kubelet service:

sudo systemctl restart kubelet

Conclusion

Setting up Kubernetes on Azure with Ubuntu VMs provides you with deep insight into Kubernetes operations and gives you full control over your cluster configuration. Though it requires more manual effort compared to cloud-managed services like AKS, it offers greater flexibility and a learning opportunity.

By following this guide and troubleshooting any issues, you'll have a robust Kubernetes setup capable of handling your containerized applications efficiently.

Part -2

Let’s configure ingress and load balancer for the cluster.