**A Step-by-Step Guide: Setting up a Kubernetes Cluster with Kubeadm on AWS EC2 Instances**

A comprehensive tutorial for beginners: Deploy a scalable Kubernetes cluster on AWS EC2 using Kubeadm with optimized instance types.

Kubernetes has emerged as a powerful platform for orchestrating containerized applications, providing scalability, fault tolerance, and ease of management. In this step-by-step guide, we will walk you through the process of setting up a Kubernetes cluster using Kubeadm, specifically on AWS EC2 instances. We will utilize an Ubuntu-based t2.medium instance for the master node and a t2.micro instance for the worker nodes.

**Prerequisites**

Before we dive into the cluster setup, make sure you have the following prerequisites in place:

1. AWS Account: Access to an AWS account with appropriate permissions to create EC2 instances.
2. SSH Key Pair: Generate an SSH key pair to securely access the EC2 instances.
3. AWS EC2 Instances: Launch an Ubuntu-based t2.medium instance for the master node and t2.micro instances for the worker nodes.
4. Security Group Configuration: Set up inbound rules to allow SSH (port 22) and Kubernetes communication (ports 6443, 2379-2380, and 10250-10252).
5. AWS CLI (Command Line Interface): Install the AWS CLI on your local machine to interact with the AWS services.

**Setting up the Master Node**

1. Launch an EC2 instance:
   1. Choose the Ubuntu 20.04 LTS AMI and the t2.medium instance type.
   2. Configure security groups to allow SSH and Kubernetes communication ports.
2. Connect to the master node:
   1. Use SSH with the generated key pair to access the EC2 instance.
3. Update the system packages:

*sudo apt-get update*

1. Install Docker:

*sudo apt-get install docker.io*

1. Start and enable Docker service:

*sudo systemctl start docker*

*sudo systemctl enable docker*

1. Add your user to the Docker group:

*sudo usermod -aG docker ubuntu*

1. Restart Docker:

*sudo systemctl restart docker*

1. Disable swap memory:

sudo swapoff -a

1. Comment out the swap entry in /etc/fstab:

*sudo sed -I ‘/ swap / s/^\(.\*\)$/#\1/g’ /etc/fstab*

1. Enable bridged traffic to pass through iptables:

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

1. Import the Kubernetes repository signing key:

*curl -s* [*https://packages*](https://packages)*.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add –*

1. Add the Kubernetes repository:

*cat <<EOF | sudo tee /etc/apt/sources.list.d/ubernetes.list*

*deb* [*https://apt*](https://apt)*.kubernetes.io/ ubernetes-xenial main*

*EOF*

1. Update the package list:

*sudo apt-get update -y*

1. Install specific versions of Kubeadm, Kubectl, and Kubelet:

*sudo apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y*

1. Initialize the Kubernetes cluster using Kubeadm:

*sudo kubeadm init*

1. Set the KUBECONFIG environment variable:

*export KUBECONFIG=/etc/ubernetes/admin.conf*

1. Deploy the flannel network plugin for pod networking:

*sudo kubectl apply -f* [*https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml*](https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml)

**Setting up the Worker Nodes**

1. Launch EC2 instances:
   1. Choose the Ubuntu 20.04 LTS AMI and the t2.micro instance type.
   2. Configure security groups to allow SSH and Kubernetes communication ports.
2. Connect to each worker node:
   1. Use SSH with the generated key pair to access the EC2 instances.
3. Follow steps 3-10 from the "Setting up the Master Node" section to configure Docker and system settings.
4. Install specific versions of Kubeadm, Kubectl, and Kubelet:

*sudo apt install kubeadm=1.20.0-00 kubectl=1.20.0-00 kubelet=1.20.0-00 -y*

1. Perform pre-flight checks to ensure the worker node is ready to join the cluster:

*sudo kubeadm reset pre-flight checks*

**Joining Worker Nodes to the Cluster**

1. On the master node, generate the join command by executing:

*sudo kubeadm token create --print-join-command*

1. Copy the generated join command.
2. On each worker node, paste and run the join command obtained from the master node don't forget to add --v=5 at the end of token.

**Verifying the Cluster**

1. Switch back to the master node.
2. Verify that all nodes have successfully joined the cluster:

*kubectl get nodes*

The output should display all the nodes in the cluster, including the master node and worker nodes.

Certainly! Based on the scenario, where we are able to access the cluster successfully as a superuser but faced issues when switching to the "ubuntu" user, you can add the following commands to solve this problem:

# Switch to the normal user

*su ubuntu*

# Set the KUBECONFIG environment variable for the user

*export KUBECONFIG=/etc/kubernetes/admin.conf*

# Copy the admin.conf file to the user's home directory

*cp /etc/kubernetes/admin.conf $HOME/*

*chown $(id -u):$(id -g) $HOME/admin.conf*

# Set the KUBECONFIG environment variable permanently for the user

*echo 'export KUBECONFIG=$HOME/admin.conf' >> $HOME/.bashrc*

Explanation:

1. Switch to the "ubuntu" user using su ubuntu to execute the following commands as that user.
2. Set the KUBECONFIG environment variable to point to the cluster configuration file /etc/kubernetes/admin.conf for the "ubuntu" user.
3. Copy the admin.conf file to the user's home directory ($HOME) and change its ownership to match the "ubuntu" user.
4. Append the export KUBECONFIG=$HOME/admin.conf command to the user's .bashrc file so that the KUBECONFIG variable is set automatically whenever the user logs in.

By adding these commands in your master instance, it will ensure that the "ubuntu" user can access the Kubernetes cluster using the copied configuration file. This will allow you to interact with the cluster without requiring superuser privileges.

Please note that after making these changes, you may need to restart the shell or log out and log back in for the changes to take effect.