### Step 2: Initialize the Master Node

1. **Initialize Kubernetes**: On the master node, run the following command to initialize Kubernetes:

sudo kubeadm init --pod-network-cidr=192.168.0.0/16

1. **Set Up kubeconfig**: After the master node initializes, set up the kubeconfig for the kubectl command.

mkdir -p $HOME/.kube

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

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

### Step 3: Install a Pod Network Add-On

1. **Choose a Network Plugin**: After the master node is initialized, you need to install a network add-on. A commonly used one is Calico. You can install it with the following command:

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

### Step 1: Get the Master IP

1. **Identify the Master Node's IP Address**:
   * On your master node, you can find the IP address by running:

ip a

My IP Address was 10.144.158.2

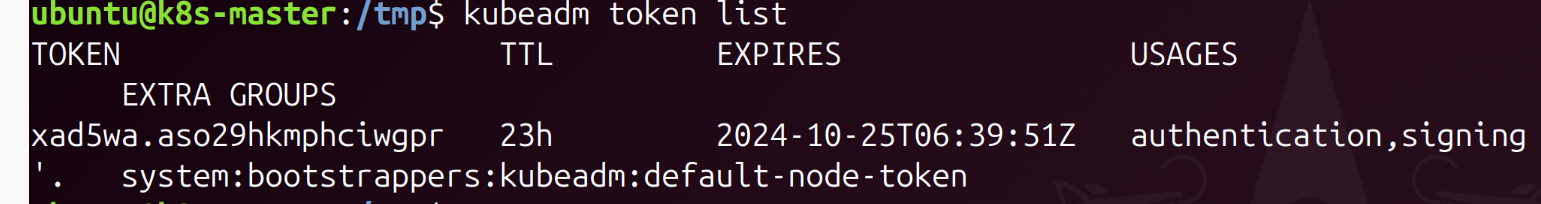
### Step 2: Get the Port

* The default port for the Kubernetes API server is 6443.

Step 3: Get the Join Token

List the Available Tokens:

kubeadm token list



Token: xad5wa.aso29hkmphciwgpr

### Step 4: Get the Discovery Token CA Cert Hash

1. **Get the CA Cert Hash**:
   * **On the Master Node** (the control plane):

* Run the following command to get the correct discovery-token-ca-cert-hash:

bash

Copy code

openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | \

openssl rsa -pubin -outform der 2>/dev/null | \

openssl dgst -sha256 -hex | sed 's/^.\* //'

* This will give you the correct hash (32 bytes long), something like:

sha256:891a78572a1017fce92fe6a46bc914688d9e3a3ed4c659fb4e28374c48e4d8f0

### Step 4: Join Worker Nodes

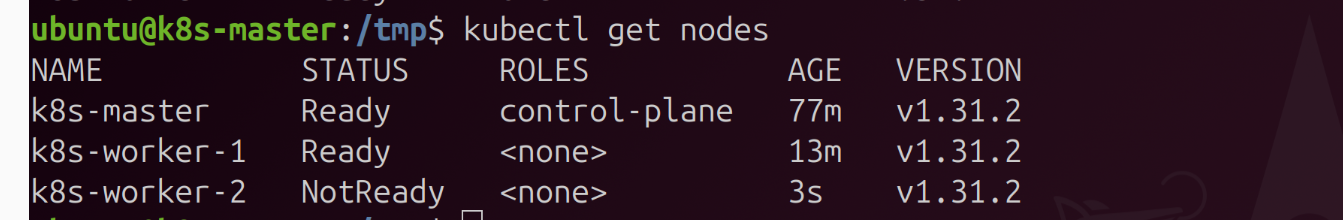
1. **Get Join Command**: After the master node initialization, you will see a command to join the worker nodes. It will look something like this:

kubeadm join <master-ip>:<port> --token <token> --discovery-token-ca-cert-hash <hash>

1. **Run Join Command on Worker Nodes**: Log into each worker node and run the join command you received from the master node.

kubeadm join 10.144.158.2:6443 --token xad5wa.aso29hkmphciwgpr --discovery-token-ca-cert-hash sha256:891a78572a1017fce92fe6a46bc914688d9e3a3ed4c659fb4e28374c48e4d8f0

After adding both workers



### ****Step 1: Set Up Pod Network (Master Node Only)****

After initializing the cluster on the master node, you need to set up the pod network so that pods can communicate across nodes.

#### On Master Node:

1. **Install a pod network (e.g., Calico):**

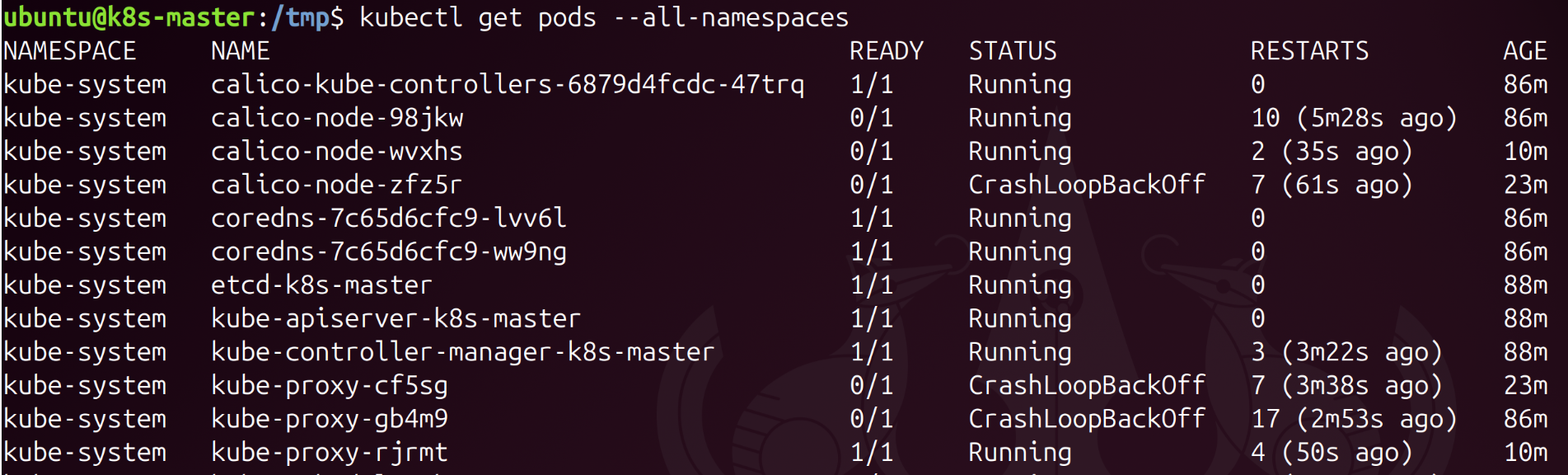
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

This configures the pod network with the CIDR you specified (192.168.0.0/16) during cluster initialization.

1. **Verify the pod network is up:**

kubectl get pods --all-namespaces

You should see the calico-node and related pods running in the output.



### ****Step 3: Deploy Test Application (Master Node Only)****

Once the worker nodes have successfully joined the cluster, you can deploy a test application on the master node to verify everything works.

#### On Master Node:

1. **Deploy a simple Nginx application:**

kubectl create deployment nginx --image=nginx

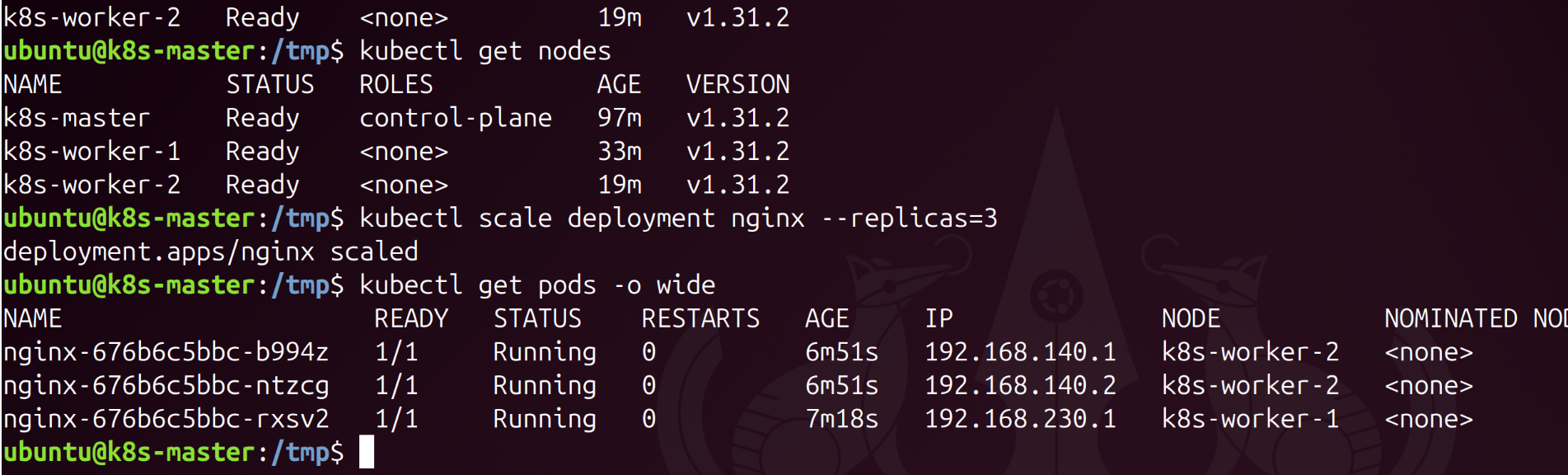
1. **Expose the Nginx deployment to access it externally:**

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

1. **Verify the Nginx service is running:**

kubectl get services

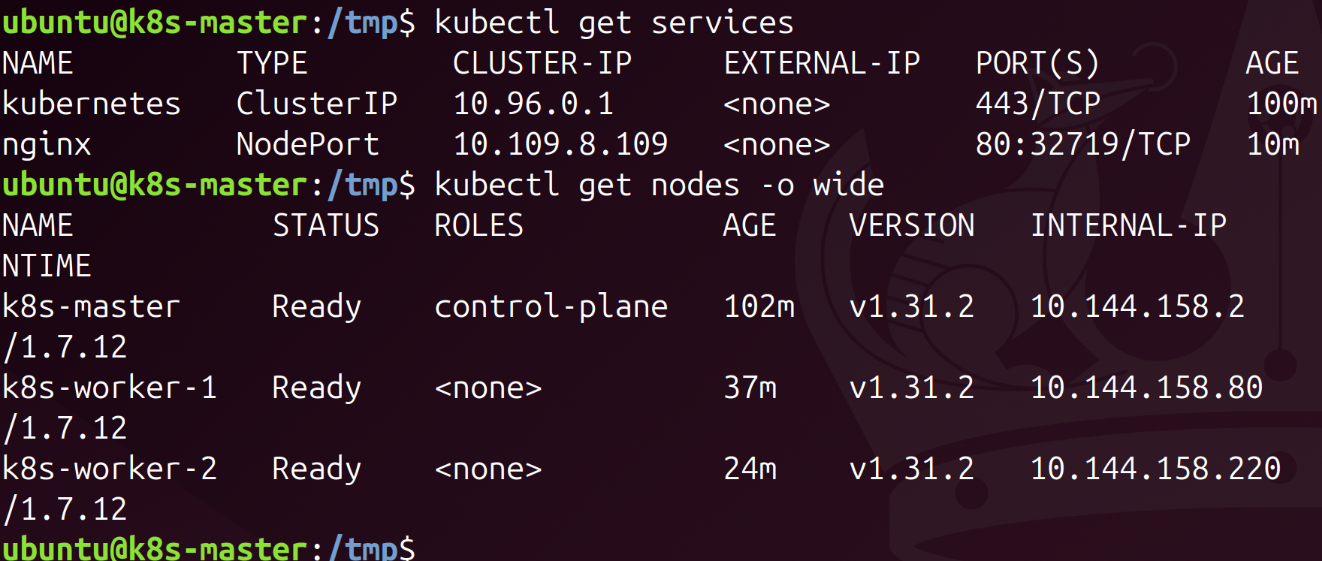
1. **Access Nginx from any browser:** Open your browser and navigate to http://<worker-node-ip>:<NodePort> (you can find the NodePort from the output of the kubectl get services command).



### ****Access the Service****

Once you have the node's IP address, you can access the nginx service by opening a browser or using curl to visit the node's IP address followed by the port number 32719:

http://<node-ip>:32719

****

Create a docker image

docker build -t ghcr.io/sujansharma07/random-forest-tuner:latest .

Login to github

echo ghp\_oqmuBLiBBAJcpiB6ZokKCmIdNTkQaz0Wf9C3 | docker login ghcr.io -u SujanSharma07 --password-stdin

Push to github packages

docker push ghcr.io/sujansharma07/random-forest-tuner:latest

#### ****Submit the Katib Experiment****

To start the experiment, apply the YAML file:

kubectl apply -f random-forest-experiment.yaml

#### 6. ****Monitor the Experiment****

Check the status of the experiment:

kubectl get experiment random-forest-experiment

To see the logs of the running trials (pods):

kubectl logs <pod-name>

#### 7. ****Access the Results****

Once the experiment is complete, you can access the results stored in the /output directory. Each trial will create a file with the hyperparameters and corresponding MSE.

If you’re using a persistent volume, you can retrieve the files directly from the mounted path:

ls /mnt/results

cat /mnt/results/result\_\*.txt

This will show the results for each hyperparameter combination.

### 1. ****Dockerfile**** (for building your Docker image)

This file contains the instructions to create the Docker image for your Random Forest model. It should include:

* Base image (e.g., python:3.8)
* Dependencies like scikit-learn
* The Python script for your hyperparameter tuning task
* Commands to run the script

### 2. ****Python Script**** (e.g., random\_forest.py)

The Python script that performs Random Forest hyperparameter tuning. This file should:

* Define the model, load the dataset, and configure the hyperparameter search
* Write the tuning results to an output file (as we discussed) so you can check the results later
* Example: random\_forest.py

### 3. ****Katib Experiment YAML File**** (e.g., random-forest-experiment.yaml)

This YAML file defines the Katib experiment for hyperparameter tuning. It should include:

* Experiment metadata (name, namespace)
* The search space for hyperparameters (like the number of trees, depth of the tree, etc.)
* The objective metric (e.g., accuracy)
* The Docker image of your script
* Number of trials, parallelism, and other Katib-specific configuration
* Example: random-forest-experiment.yaml

### 4. ****Kubernetes Job YAML (Optional)**** (e.g., random-forest-job.yaml)

If you're running your hyperparameter tuning task as a Kubernetes job without Katib, you may have a Kubernetes Job manifest. This file will:

* Specify the image to be run
* Configure resource limits (CPU, memory)
* Define volumes if needed for storing results or data

### 5. ****Kubernetes Cluster Configuration**** (Optional)

If you set up specific configurations like namespace or secret files for Docker registry access (e.g., if using GitHub Container Registry for Docker images), you may have:

* namespace.yaml: For creating a specific namespace to run the job or experiment.
* secret.yaml: For storing Docker registry credentials if private registry access is needed.

### 6. ****Output Files**** (Generated at Runtime)

As part of the experiment, your Python script should write output to a file to capture the results of different hyperparameter combinations. You should decide where to store these files, and they will be generated during the job's execution:

* **Tuning result file**: This file contains the results of each trial, such as the accuracy of each hyperparameter combination.
  + Example: results.txt or best\_params.json.

### 1. ****Kubernetes Dashboard****

The Kubernetes Dashboard is a general-purpose web-based UI for Kubernetes clusters. It allows you to manage applications running in your cluster, troubleshoot them, and view logs. You can also use it to manage Kubernetes resources such as deployments, jobs, and services.

#### How to Install and Access Kubernetes Dashboard:

1. **Install the Dashboard**: Run the following command to deploy the Kubernetes Dashboard:

bash

Copy code

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.6.0/aio/deploy/recommended.yaml

1. **Access the Dashboard**: To access the dashboard, you'll need to create a service account and a cluster role binding:

bash

Copy code

kubectl create serviceaccount dashboard-admin -n kubernetes-dashboard

kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serviceaccount=kubernetes-dashboard:dashboard-admin

Then get the token to log in:

bash

Copy code

kubectl get secret -n kubernetes-dashboard $(kubectl get serviceaccount dashboard-admin -n kubernetes-dashboard -o jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 --decode

Finally, access the Dashboard by running:

bash

Copy code

kubectl proxy

Then navigate to http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/ in your web browser.