NVIDIA NIM on Charmed Kubeflow Tutorial

This tutorial describes the complete steps to deploy a NIM (NVIDIA Inference Microservice) on Charmed Kubeflow from Canonical.

A NIM is a containerized inference microservice distributed from NVIDIA’s private container registry - NGC (NVIDIA GPU Cloud).

By the end of this tutorial, you will be able to:

1. Deploy MicroK8s, a small, fast, single-package Kubernetes for datacenters and the edge.

2. Deploy Charmed Kubeflow, which enables developing and deploying machine learning models at any scale.

3. Deploy a NIM and serve a model with KServe, an addon component of Kubeflow.

This tutorial uses Kubeflow v1.8, Kubernetes v1.28, and Juju v3.1. Check the supported version combinations here:

<https://charmed-kubeflow.io/docs/supported-versions>

A GPU that is compatible with the model downloaded from NGC is required. Refer to the model details on NGC for further information. This tutorial is tested on a machine with an NVIDIA A100 GPU.

# Deploy Ubuntu Server 22.04 LTS

~~# Deploy Ubuntu Server 24.04 LTS~~

<https://ubuntu.com/tutorials/install-ubuntu-server>

# Update system

sudo apt update && sudo apt upgrade -y

# Install NVIDIA GPU driver

sudo apt install nvidia-headless-535-server nvidia-utils-535-server -y

# Reboot system and check NVIDIA GPU devices

sudo reboot

nvidia-smi

# Install MicroK8s

sudo snap install microk8s --channel=1.28/stable --classic

~~sudo snap install microk8s --channel=1.29/stable --classic~~

# Add the current user to the microk8s group to avoid having to use sudo for every microk8s command

sudo usermod -a -G microk8s $USER

newgrp microk8s

# Enable MicroK8s add-ons needed to run Charmed Kubeflow

IP=$(hostname -I | awk '{print $1}')

microk8s enable dns hostpath-storage ingress gpu metallb:"$IP-$IP"

microk8s enable dns hostpath-storage nvidia ingress metallb:10.64.140.43-10.64.140.49

# Check MicroK8s status until the output shows "microk8s is running" and the add-ons installed are listed under "enabled"

microk8s status --wait-ready

# Add an alias for omitting microk8s when running kubectl commands

alias kubectl='microk8s kubectl'

echo "alias kubectl='microk8s kubectl'" >> ~/.bash\_aliases

# Install Juju

~~sudo snap install juju --channel=3.1/stable~~

sudo snap install juju --channel=3.5/stable

# Configure MicroK8s to work properly with Juju

microk8s config | juju add-k8s microk8s-1 --client

Note: Command “microk8s config” retrieves the client’s Kubernetes config which is then registered to Juju Kubernetes endpoints.

# Deploy Juju controller to MicroK8s

juju bootstrap microk8s-1

# Add model for Kubeflow

juju add-model kubeflow

# Deploy Charmed Kubeflow

juju deploy kubeflow --trust --channel=1.8/stable

~~juju deploy kubeflow --trust --channel=1.9/stable~~

# Check juju status until all apps, except dex-auth, istio-pilot, and oidc-gatekeeper, become active

watch -c 'juju status --color | grep -E "blocked|error|maintenance|waiting|App|Unit"'

~~# Configure oidc-gatekeeper with the ingress gateway IP~~

~~IP=$(microk8s kubectl -n kubeflow get svc istio-ingressgateway-workload -o jsonpath='{.status.loadBalancer.ingress[0].ip}')~~

~~juju config oidc-gatekeeper public-url=http://$IP~~

~~# Check juju status until all apps become active~~

~~watch -c 'juju status --color | grep -E "blocked|error|maintenance|waiting|App|Unit"'~~

Create an account at https://ngc.nvidia.com/signin and create an API key at https://org.ngc.nvidia.com/setup/api-key

Note: You must have NIM access on NGC.

# Set an environment variable for the API key

export NGC\_CLI\_API\_KEY=<key>

# Create Kubernetes secret with the NGC API key to download NIMs from NGC private Docker registry

kubectl create secret docker-registry ngc-docker-secret \

--docker-server=nvcr.io\

--docker-username='$oauthtoken'\

--docker-password=$NGC\_CLI\_API\_KEY

# Create Kubernetes secret with the NGC API key to launch NIMs

kubectl create secret generic ngc-nim-secret --from-literal=NGC\_CLI\_API\_KEY=$NGC\_CLI\_API\_KEY

# Install NVIDIA NGC CLI

~~sudo apt install unzip~~

~~wget --content-disposition https://api.ngc.nvidia.com/v2/resources/nvidia/ngc-apps/ngc\_cli/versions/3.43.0/files/ngccli\_linux.zip -O ngccli\_linux.zip && unzip ngccli\_linux.zip~~

wget --content-disposition https://api.ngc.nvidia.com/v2/resources/nvidia/ngc-apps/ngc\_cli/versions/3.52.0/files/ngccli\_linux.zip -O ngccli\_linux.zip && sudo apt install unzip && unzip ngccli\_linux.zip

echo "export PATH=\"\$PATH:$(pwd)/ngc-cli\"" >> ~/.bash\_profile && source ~/.bash\_profile

# Configure NGC CLI client. Enter API key, enter org, leave everything else as default

ngc config set

# Download the model

~~ngc registry model download-version "mphexwv2ysej/meta-llama3-8b-instruct:0515-db4a5074-trtllm10-1xa100-fp16"~~

ngc registry model download-version "nvidia/llama3-8b-instruct:1.0"

sudo mkdir /mnt/nim

~~sudo mv meta-llama3-8b-instruct\_v0515-db4a5074-trtllm10-1xa100-fp16 /mnt/nim/~~

sudo mv llama3-8b-instruct\_v1.0 /mnt/nim/

~~# Enable the NodeSelector feature of KServe to allow a NIM to request different GPU types~~

~~kubectl patch configmap config-features -n knative-serving --type merge -p '{"data":{"kubernetes.podspec-nodeselector":"enabled"}}'~~

# Create a PVC called nim-pvc in the cluster

cat > nim-model-volume.yaml << EOL

apiVersion: v1

kind: PersistentVolume

metadata:

name: nim-pv

spec:

capacity:

storage: 100Gi

volumeMode: Filesystem

accessModes:

- ReadWriteMany

persistentVolumeReclaimPolicy: Retain

storageClassName: microk8s-hostpath

local:

path: /mnt/nim

nodeAffinity:

required:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

- `hostname`

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: nim-pvc

spec:

accessModes:

- ReadWriteMany

storageClassName: microk8s-hostpath

resources:

requests:

storage: 100Gi

EOL

kubectl create -f nim-model-volume.yaml

# Create ClusterServingRuntime

cat > cluster-serving-runtime.yaml << EOL

apiVersion: serving.kserve.io/v1alpha1

kind: ClusterServingRuntime

metadata:

name: nim-meta-llama3-8b-instruct-24.05.rc11

spec:

annotations:

prometheus.kserve.io/path: /metrics

prometheus.kserve.io/port: "8000"

serving.kserve.io/enable-metric-aggregation: "true"

serving.kserve.io/enable-prometheus-scraping: "true"

containers:

- env:

- name: NIM\_CACHE\_PATH

value: /tmp

- name: NGC\_API\_KEY

valueFrom:

secretKeyRef:

name: ngc-nim-secret

key: NGC\_CLI\_API\_KEY

~~image: nvcr.io/mphexwv2ysej/meta-llama3-8b-instruct:24.05.rc11~~

image: nvcr.io/nim/meta/llama3-8b-instruct:1.0.0

name: kserve-container

ports:

- containerPort: 8000

protocol: TCP

resources:

limits:

cpu: "1"

memory: 16Gi

requests:

cpu: "1"

memory: 16Gi

volumeMounts:

- mountPath: /dev/shm

name: dshm

imagePullSecrets:

- name: ngc-docker-secret

protocolVersions:

- v2

- grpc-v2

supportedModelFormats:

- autoSelect: true

name: nim-meta-llama3-8b-instruct

priority: 1

version: "24.05"

volumes:

- emptyDir:

medium: Memory

sizeLimit: 16Gi

name: dshm

EOL

kubectl create -f cluster-serving-runtime.yaml

# Create InferenceService

cat > inference-service.yaml << EOL

apiVersion: serving.kserve.io/v1beta1

kind: InferenceService

metadata:

annotations:

autoscaling.knative.dev/target: "10"

name: nim-meta-llama3-8b-instruct-1xgpu

spec:

predictor:

minReplicas: 1

model:

modelFormat:

name: nim-meta-llama3-8b-instruct

resources:

limits:

nvidia.com/gpu: "1"

requests:

nvidia.com/gpu: "1"

runtime: nim-meta-llama3-8b-instruct-24.05.rc11

storageUri: pvc://nim-pvc/

EOL

kubectl create -f inference-service.yaml

# Watch the status of the pod created until it becomes ready

kubectl describe pod nim-meta-llama3-8b-instruct-1xgpu

Note: It can take a while for the pod to become functional. The warning message "Readiness probe failed" can be ignored. Continue with the instructions below and start testing queries.

# Get the IP address of the private predictor

KSERVE=$(kubectl get svc |grep private |awk '{print $3}')

# Validate that the NIM is running by posting a query against the KServe endpoint

curl http://$KSERVE/v1/chat/completions -H "Content-Type: application/json" -d '{

"model": "meta-llama3-8b-instruct",

"messages": [{"role":"user","content":"What is KServe?"}]

}'

curl http://$KSERVE/v1/chat/completions -H "Content-Type: application/json" -d '{

"model": "llama3-8b-instruct",

"messages": [{"role":"user","content":"What is KServe?"}]

}'

# Cleanup

sudo snap remove --purge juju

sudo snap remove --purge microk8s

sudo rm -rf /home/$USER/.local/share/juju

rm -rf ~/.kube/

Gokhan Cetinkaya

Alliances Engineer

gokhan.cetinkaya@canonical.com

www.linkedin.com/in/gcetinkaya

References:

<https://github.com/NVIDIA/nim-deploy/tree/main/kserve>