**Installing watsonx.ai on ROKS**

# Introduction

This document provides end-to-end steps for setting up IBM watsonx.ai and LLM on ROKS (Red Hat OpenShift on IBM Cloud), starting from provisioning a ROKS cluster all the way to deploying one or more large language models (LLMs).

The official documentation covers a wide range of options, but not all of them are applicable or required to this specific setup. This guide focuses only on the relevant steps and makes the following assumptions:

* The ROKS cluster has internet access (no private registry required).
* OpenShift AI is installed through IBM Cloud.
* OpenShift Data Foundation (ODF) is installed through IBM Cloud.
* The Software Hub version is **5.2.1**.
* The OpenShift AI add-on currently has issues installing the GPU Operator.
* You have Administrator permission for your IBM account
* You use IBM Cloud Gen 2 infrastructure (none-bare metal)
* Your client machine is Mac (running z-shell)

At a high level, the installation process includes:

1. Provisioning Cloud infrastructure
2. Provisioning a ROKS cluster and storage
3. Setting up client machine
4. Preparing cluster
5. Installing shared components
6. Installing Software Hub
7. Deploying watsonx.ai
8. Deploying LLMs

# Cloud Infrastructure Provisioning

## Step 1: Provision a VPC in IBM Cloud

1. Log in to the IBM Cloud Console.
2. From the main menu, go to **Infrastructure**→ **Network**→ **VPCs**
3. Click **Create** VPC.

A screenshot of a computer

AI-generated content may be incorrect.

1. Enter the following details:

* **Name**: Provide a meaningful name (e.g., watsonx-vpc).
* **Resource group**: Select an existing resource group or create a new one.
* **Location**: Choose the region where you want to deploy your cluster (e.g., us-south).
* **Default security group rules**: Leave the defaults or adjust if you need custom ingress/egress rules.

1. Under **Address prefixes**, leave it as **Automatically assign**, unless you need to specify a custom CIDR block.
2. Under **Subnets**, you can either:

* Let IBM Cloud automatically create subnets for each zone, or
* Manually add subnets (recommended if you want more control).

1. Review your configuration and click **Create VPC**.

A screenshot of a computer

AI-generated content may be incorrect.

## Step 2: Create a Load Balancer in IBM Cloud

1. In the IBM Cloud Console, open the navigation menu and go to **Infrastructure** → **Network**→ **Load balancers**.
2. Click **Create**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Choose the **Application Load Balancer** (ALB) load balancer type.
2. Enter the following details:

* **Name**: e.g., watsonx-lb.
* **Resource group**: Use the same resource group as your VPC.
* **Region**: Match the region of your VPC.
* **VPC**: Select the VPC you created earlier.
* **Subnets**: Select one subnet per availability zone.

1. Choose **Public** as access type and **DNS** type
2. Review the configuration and click **Create load balancer**.

A screenshot of a computer

AI-generated content may be incorrect.

# ROKS Cluster Provisioning and Storage

## Step 1: Create base OpenShift Cluster

1. In the IBM Cloud Console, open the navigation menu and go to **Containers → Clusters**.
2. Click **Create cluster**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Select cluster type: **Red Hat OpenShift**.
2. Select infrastructure: **VPC**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Configure cluster basics.

* **VPC**: Select the VPC you created earlier.
* **Region**: Use the same region as your VPC.
* **Zone(s)**: Select one or more zones. Multi-zone is recommended for high availability.
* **Subnets**: Choose subnets in your selected zone(s).
* **OpenShift version**: Use **4.17.36** (**Note**: version 4.18 currently causes issues).

A screenshot of a computer

AI-generated content may be incorrect.

1. Configure worker pool.

* **Machine flavor**: Select a VM profile (for example, bx2.16x64 as the base). You can add a GPU-enabled worker pool later.
* **Worker node count per zone**: At least 3 nodes are recommended.
* **Operating system**: Use the default (RHCOS).
* **Worker pool encryption**: Optional.

A screenshot of a computer

AI-generated content may be incorrect.

1. Configure network settings.

* Enable **Both private & public endpoints** unless you require a private-only cluster.
* **Network Security:** Disable **Outbound traffic protection** for online installation.
* If using a private cluster, ensure **VPN** or **Transit Gateway access** is configured.

**A screenshot of a computer

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

1. Configure cluster details.

* **Name**: Enter a meaningful name (for example, watsonx-roks).
* **Resource group**: Use the same resource group as your VPC.

1. Configure integrations.

* Leave all integration options at their default settings.

1. Review the configuration and click **Create cluster**.

Once the cluster provisioning begins, it will take around **20–40 minutes** to complete. You can monitor progress from the **Clusters** dashboard.

**A screenshot of a computer

AI-generated content may be incorrect.**

## Step 2: Create Storage Worker Nodes

1. In the IBM Cloud Console, open the navigation menu and go to **Containers → Clusters**, then select the cluster you created.
2. Select Worder pools on the left menu and **click Add**

A screenshot of a computer

AI-generated content may be incorrect.

1. Configure the worker pool.

* **Name**: Enter a descriptive name (e.g., storage-worker-pool).
* **Worker node count per zone**: Specify the number of worker nodes for each zone. Consider at least 1–3 nodes depending on workload requirements.
* **Worker Node flavor**: Choose a VM profile appropriate for storage (e.g., flavor like bx2d.16x64 works well for storge nodes).
* **Worker pool encryption**: Optional, enable if required for compliance/security.

1. Review the configuration and click **Create**.

A screenshot of a computer

AI-generated content may be incorrect.

## Step 3: Create ODF Storage

1. In the IBM Cloud Console, open the navigation menu and go to **Containers → Clusters**, then select the cluster you created.
2. Scroll down to the **Add-ons** section, locate **OpenShift Data Foundation (ODF)**, and click **Install**.
3. Leave the **version** and **plan** settings as default.
4. **Storage type**: Select **Remote provisioning**.
5. Configure backing storage.

* **OSD Storage class name**: Use the default.
* **Resource profile**: Select **Performance**.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Capacity and worker nodes**.

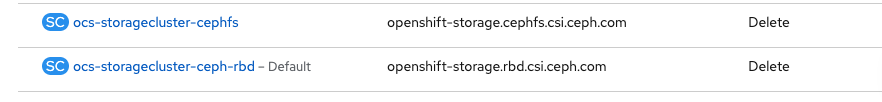
* **OSD Pod size**: Set the usable storage size.
* **Number of OSD Disks Required**: Set to **1**.
* **Worker pools**: Select the worker pool for your storage nodes.
* Enable **Taint nodes** and **Use Ceph RADOS device**.

1. Review your configuration and click **Install**.

A screenshot of a computer

AI-generated content may be incorrect.

The ODF installation will take about 5-10 minutes. You can check if the storage class to make sure the installation is successful.



## Step 3: Adding GPU Nodes

NOTE: GPU resources are costly, so it’s best to make sure your ROKS environment is stable before adding them.

Adding GPU worker nodes is similar to adding storage nodes. You create a new worker pool and select a machine flavor that includes the GPU resources you need.

The worker node flavor must meet the requirements of the foundation model you plan to deploy. For details, see the [Foundation Models](https://www.ibm.com/docs/en/software-hub/5.2.x?topic=install-foundation-models)section in the IBM watsonx.ai documentation.

**Note:** The storage requirements shown in the model specification tables in the documentation refer to ephemeral storage, not persistent storage. IBM Cloud worker nodes provide 100 GB of ephemeral storage by default. If your model requires more, you need to attach additional secondary storage. This extra storage becomes part of the node’s ephemeral storage and must be planned and configured when creating the GPU worker pool to ensure the model deploys successfully.

A screenshot of a computer

AI-generated content may be incorrect.

# Setup Client Machine

Before installing Software Hub, you must prepare your local Mac environment to connect to your IBM Cloud ROKS cluster and run the installation commands.

## Step 1: Install IBM Cloud CLI

1. Open a terminal window and run the following command.

curl -fsSL https://clis.cloud.ibm.com/install/osx | sh

1. Verify installation.

ibmcloud –version

1. IBM Cloud CLI uses plug-ins to extend functionality. The following are the most common ones.

* Container and Kubernetes

ibmcloud plugin install container-service

ibmcloud plugin install container-registry

* VPC Infrastructure

ibmcloud plugin install vpc-infrastructure

## Step 2: Install OpenShift CLI (oc)

1. Download and install the latest OpenShift CLI.

curl -L https://mirror.openshift.com/pub/openshift-v4/clients/ocp/latest/openshift-client-mac.tar.gz -o oc.tar.gz

tar -xvzf oc.tar.gz

sudo mv oc /usr/local/bin/

sudo mv kubectl /usr/local/bin/

1. Verify installation.

oc version

kubectl version –client

## Step 3: Install Podman

1. The easiest way is with **Homebrew**.

brew install podman

1. Initialize Podman by running the following to set up the Podman machine (it runs inside a lightweight VM on macOS).

podman machine init

1. Start the Podman machine

podman machine start

1. Test the installation

podman run hello-world

## Step 4: Install CPD CLI (Cloud Pak for Data CLI)

1. Download and install CPD CLI.

curl -LO https://github.com/IBM/cpd-cli/releases/download/v14.2.1/cpd-cli-darwin-EE-14.2.1.tgz

1. Download and install CPD CLI (Mac ARM).

curl -LO <https://github.com/IBM/cpd-cli/releases/latest/download/cpd-cli-darwin-arm64.tar.gz>

1. After downloading, extract and install by running the following commands.

tar -xvzf cpd-cli-darwin-EE-14.2.1.tgz

sudo mv cpd-cli /usr/local/bin/

echo 'export PATH=/usr/local/bin/cpd-cli:$PATH' >> ~/.zshrc

source ~/.zshrc

1. Grant execution permission on Mac.

* Open File Finder and navigate to /usr/local/bin/cpd-cli directory
* Double-click on cpd-cli
* Navigate to /usr/local/bin/cpd-cli/plugins/lib/Darwin
* Double-click (Open) every single plugin file

1. Verify installation:

cpd-cli version

## Step 5: Target (setup local context) your ROKS cluster

1. Log in to IBM Cloud

ibmcloud login –sso

* You’ll be prompted to open a browser and paste a one-time code for authentication.
* Then select your user account.

1. List available clusters.

ibmcloud ks clusters

1. Configure your local CLI for the cluster.

ibmcloud ks cluster config --cluster <CLUSTER\_NAME\_OR\_ID>

## Steps 6: Setup environment variable

1. Retrieve IBM entitlement key from

<https://myibm.ibm.com/products-services/containerlibrary>

1. Copy the following example to a text editor on your local file system

* Uncomment all orange color lines.
* Provide specific values for place holder values “<>”.

#===============================================================================

# IBM Software Hub installation variables

#===============================================================================

# ------------------------------------------------------------------------------

# Client workstation

# ------------------------------------------------------------------------------

# Set the following variables if you want to override the default behavior of the IBM Software Hub CLI.

#

# To export these variables, you must uncomment each command in this section.

export CPD\_CLI\_MANAGE\_WORKSPACE=<enter a fully qualified directory>

# export OLM\_UTILS\_LAUNCH\_ARGS=<enter launch arguments>

# ------------------------------------------------------------------------------

# Cluster

# ------------------------------------------------------------------------------

export OCP\_URL=<enter your Red Hat OpenShift Container Platform URL>

export OPENSHIFT\_TYPE=roks

export IMAGE\_ARCH=amd64

# export OCP\_USERNAME=<enter your username>

# export OCP\_PASSWORD=<enter your password>

export OCP\_TOKEN=<enter your token>

export SERVER\_ARGUMENTS="--server=$OCP\_URL"

# export LOGIN\_ARGUMENTS="--username=${OCP\_USERNAME} --password=${OCP\_PASSWORD}"

export LOGIN\_ARGUMENTS="--token=$OCP\_TOKEN"

export CPDM\_OC\_LOGIN="cpd-cli manage login-to-ocp $SERVER\_ARGUMENTS ${LOGIN\_ARGUMENTS}"

export OC\_LOGIN="oc login ${SERVER\_ARGUMENTS} $LOGIN\_ARGUMENTS"

# ------------------------------------------------------------------------------

# Proxy server

# ------------------------------------------------------------------------------

# export PROXY\_HOST=<enter your proxy server hostname>

# export PROXY\_PORT=<enter your proxy server port number>

# export PROXY\_USER=<enter your proxy server username>

# export PROXY\_PASSWORD=<enter your proxy server password>

# export NO\_PROXY\_LIST=<a comma-separated list of domain names>

# ------------------------------------------------------------------------------

# Projects

# ------------------------------------------------------------------------------

#export PROJECT\_CERT\_MANAGER=<enter your certificate manager project>

export PROJECT\_LICENSE\_SERVICE=ibm-licensing

export PROJECT\_SCHEDULING\_SERVICE=ibm-cpd-scheduler

# export PROJECT\_IBM\_EVENTS=<enter your IBM Events Operator project>

# export PROJECT\_PRIVILEGED\_MONITORING\_SERVICE=<enter your privileged monitoring service project>

export PROJECT\_CPD\_INST\_OPERATORS=cpd-operator

export PROJECT\_CPD\_INST\_OPERANDS=cpd-operand

# export PROJECT\_CPD\_INSTANCE\_TETHERED=<enter your tethered project>

# export PROJECT\_CPD\_INSTANCE\_TETHERED\_LIST=<a comma-separated list of tethered projects>

# ------------------------------------------------------------------------------

# Storage

# ------------------------------------------------------------------------------

export STG\_CLASS\_BLOCK=ocs-storagecluster-ceph-rbd

export STG\_CLASS\_FILE=ocs-storagecluster-cephfs

# ------------------------------------------------------------------------------

# IBM Entitled Registry

# ------------------------------------------------------------------------------

export IBM\_ENTITLEMENT\_KEY=<enter your IBM entitlement API key>

# ------------------------------------------------------------------------------

# Private container registry

# ------------------------------------------------------------------------------

# Set the following variables if you mirror images to a private container registry.

#

# To export these variables, you must uncomment each command in this section.

# export PRIVATE\_REGISTRY\_LOCATION=<enter the location of your private container registry>

# export PRIVATE\_REGISTRY\_PUSH\_USER=<enter the username of a user that can push to the registry>

# export PRIVATE\_REGISTRY\_PUSH\_PASSWORD=<enter the password of the user that can push to the registry>

# export PRIVATE\_REGISTRY\_PULL\_USER=<enter the username of a user that can pull from the registry>

# export PRIVATE\_REGISTRY\_PULL\_PASSWORD=<enter the password of the user that can pull from the registry>

# ------------------------------------------------------------------------------

# IBM Software Hub version

# ------------------------------------------------------------------------------

export VERSION=5.2.1

# ------------------------------------------------------------------------------

# Components

# ------------------------------------------------------------------------------

export COMPONENTS=ibm-licensing,scheduler,cpfs,cpd\_platform

# export COMPONENTS\_TO\_SKIP=<component-ID-1>,<component-ID-2>

# export IMAGE\_GROUPS=<image-group-1>,<image-group-2>

# Preparing Cluster

## Step 1: Create namepsaces

oc new-project $PROJECT\_LICENSE\_SERVICE

oc new-project $PROJECT\_SCHEDULING\_SERVICE

oc new-project $PROJECT\_CPD\_INST\_OPERATORS

oc new-project $PROJECT\_CPD\_INST\_OPERANDS

## Step 2: Create global pull secrete

1. Login to the cluster

eval $CPDM\_OC\_LOGIN

1. Add global pull secret

cpd-cli manage add-icr-cred-to-global-pull-secret \

--entitled\_registry\_key=$IBM\_ENTITLEMENT\_KEY

oc create secret docker-registry docker-auth-secret \

--docker-server=cp.icr.io \

--docker-username=cp \

--docker-password=$IBM\_ENTITLEMENT\_KEY \

--namespace kube-system

1. Populate global secrete to all the worker nodes on ROKS.

* Create a daemonset yaml file (gs-daemonset.yaml) using the code below.

apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: update-docker-config

  namespace: kube-system

  labels:

    app: update-docker-config

spec:

  selector:

    matchLabels:

      name: update-docker-config

  template:

    metadata:

      labels:

        name: update-docker-config

    spec:

      initContainers:

      - name: updater

        image: registry.access.redhat.com/ubi8/ubi-minimal:latest  # Use UBI instead of Alpine

        imagePullPolicy: IfNotPresent

        command: ["/bin/bash", "-c"]

        args:

        - |

          # Install jq

          microdnf update -y && microdnf install -y jq

          # Set config path

          CONFIG\_PATH="/host/var/lib/kubelet"

          echo "Using config path: $CONFIG\_PATH"

          # Check if config exists

          if [[ ! -f "$CONFIG\_PATH/config.json" ]]; then

            echo "ERROR: config.json not found at $CONFIG\_PATH/config.json"

            ls -la $CONFIG\_PATH/

            exit 1

          fi

          # Backup current config

          echo "Backing up config.json..."

          cp -v "$CONFIG\_PATH/config.json" "$CONFIG\_PATH/config.json.bak"

          # Merge secret with config.json

          echo "Merging secret with config.json..."

          cat /auth/.dockerconfigjson | jq .

          if jq -s '.[0] \* .[1]' "$CONFIG\_PATH/config.json" /auth/.dockerconfigjson > "$CONFIG\_PATH/config.tmp"; then

            echo "Merge successful"

            mv -v "$CONFIG\_PATH/config.tmp" "$CONFIG\_PATH/config.json"

            echo "Config updated successfully"

          else

            echo "ERROR: jq merge failed"

            exit 1

          fi

          echo "Update completed successfully."

        securityContext:

          privileged: true

          runAsUser: 0

        volumeMounts:

        - name: docker-auth-secret

          mountPath: /auth

          readOnly: true

        - name: host

          mountPath: /host

          readOnly: false

      containers:

      - name: sleep

        image: registry.access.redhat.com/ubi8/ubi-minimal:latest

        command: ["sleep", "infinity"]

        resources:

          requests:

            cpu: "10m"

            memory: "32Mi"

        volumeMounts:

        - name: host

          mountPath: /host

          readOnly: true

      volumes:

      - name: docker-auth-secret

        secret:

          secretName: docker-auth-secret

      - name: host

        hostPath:

          path: /

          type: Directory

* Deploy the yaml file.

oc apply -f gs-daemonset.yaml

1. Verify global secret

* Get work node list.

oc get nodes

* Verify global secrete for cp.icr.io in worker nodes.

oc debug node/<worker node> -- chroot /host cat /var/lib/kubelet/config.json | jq '.auths["cp.icr.io"]'

You should see similar result, which has cp user and its password, to below.

Starting pod/10240072-debug-4pss7 ...

To use host binaries, run `chroot /host`

{

Removing debug pod ...

"auth": "Y3A6ZXlKMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKSVV6STFOaUo5LmV5SnBjM01pT2lKSlFrMGdUV0Z5YTJWMGNHeGhZMlVpTENKcFlYUWlPakUzTURRNE1UazFOVFlzSW1wMGFTSTZJakEzTXpkbE9EVXpaRGhqWXpRNFlqQmhOamt4TkRKaE5qYzBZell4TVdaaEluMC5IQ2cwVDZPeW0zZmUwVEFNXy1vSG9FYXp4c0kyQnhaRElQTHhOclZyM2Vj",

"username": "cp",

"password": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJJQk0gTWFya2V0cGxhY2UiLCJpYXQiOjE3MDQ4MTk1NTYsImp0aSI6IjA3MzdlODUzZDhjYzQ4YjBhNjkxNDJhNjc0YzYxMWZhIn0.HCg0T6Oym3fe0TAM\_-oHoEazxsI2BxZDIPLxNrVr3ec"

}

**Note:** If you need to update the global secret, delete the existing one and recreate it. You will also need to delete the DaemonSet instances, so they will restart and propagate the new secret to the worker nodes.

**The following provide the commands.**

kubectl delete secret docker-auth-secret --namespace kube-system

kubectl delete pods -l name=update-docker-config --namespace=kube-system

## Step 3: Change load balancer timeout setting for watsonx.ai

oc patch svc router-default \

--namespace=openshift-ingress \

--type=merge \

--patch '{"metadata": {"annotations": {"service.kubernetes.io/ibm-load-balancer-cloud-provider-vpc-idle-connection-timeout": "600"}}}'

# Install Shared Components

## Step 1: Install Red Hat OpenShift Cert Manager

oc apply -f - <<EOF

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: cert-manager-operator

namespace: openshift-operators

spec:

channel: stable-v1.17

name: cert-manager

source: redhat-operators

sourceNamespace: openshift-marketplace

EOF

## Step 2: Install license service

1. Log in to the cluster.

eval $CPDM\_OC\_LOGIN

1. Deploy the license service operator.

cpd-cli manage apply-cluster-components \

--release=$VERSION \

--license\_acceptance=true \

--licensing\_ns=$PROJECT\_LICENSE\_SERVICE

Wait for the cpd-cli to return the following message before proceeding to the next step.

[SUCCESS]... The apply-cluster-components command ran successfully.

## Step 3: Install scheduling service

1. Deploy the scheduling service operator.

cpd-cli manage apply-scheduler \

--release=$VERSION \

--license\_acceptance=true \

--scheduler\_ns=$PROJECT\_SCHEDULING\_SERVICE

**Note:** This command will not complete until you patch the service account with the cp.icr.io pull secrete due to Scheuler metric pod uses service account and cannot use the global pull secrete. The service account needs to be patched to be associated with the pull secrete from its own namespace.

1. Patch service account by running the following command in another terminal window.

* Create pull secrete in the same namespace.

oc create secret docker-registry docker-auth-secret \

  --docker-server=cp.icr.io \

  --docker-username=cp \

  --docker-password="$IBM\_ENTITLEMENT\_KEY" \

  --namespace=ibm-cpd-scheduler

* Patch the service account to be associated with the pull secret.

oc patch serviceaccount ibm-cpd-scheduler-metrics-sa -n ibm-cpd-scheduler -p '{"imagePullSecrets": [{"name": "ibm-cpd-scheduler-metrics-sa-dockercfg-wcvhr"}, {"name": "docker-auth-secret"}]}'

## Step 4: Install NFD operator.

1. Create a yaml file based on below code and name it nfd\_operator\_install.yaml.

apiVersion: v1

kind: Namespace

metadata:

name: openshift-nfd

labels:

name: openshift-nfd

openshift.io/cluster-monitoring: "true"

---

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

metadata:

name: openshift-nfd

namespace: openshift-nfd

spec:

targetNamespaces:

- openshift-nfd

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: nfd

namespace: openshift-nfd

spec:

channel: stable

installPlanApproval: Automatic

name: nfd

source: redhat-operators

sourceNamespace: openshift-marketplace

1. Deploy the yaml file.

oc apply -f nfd\_operator\_install.yaml

1. Verify the Node Feature Discovery Operator is running.

oc get pods -n openshift-nfd

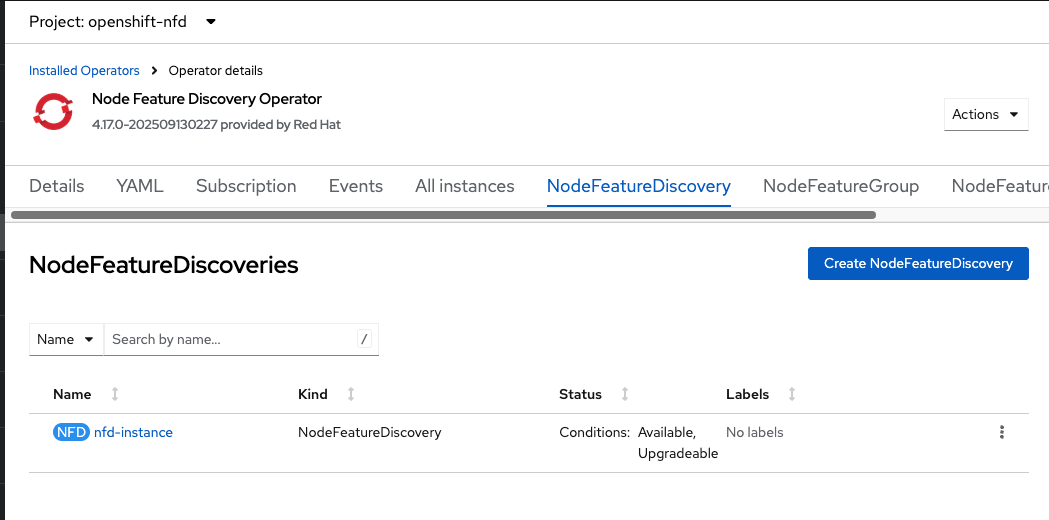
You should see result similar to the below.

NAME READY STATUS RESTARTS AGE

nfd-controller-manager-7f86ccfb58-nqgxm 2/2 Running 0 11m

1. Create a Node Feature Discovery instance.

* Find the NFD Operator in namespace openshift-nfd
* Click on the operator
* Click on **NodeFeatureDiscovery** tab
* Click on **Create NodeFeatureDiscovery** to create an instance.



1. Verify the GPU device (**pci-10de**) is discovered on the GPU node

oc describe node | egrep 'Roles|pci' | grep pci-10de

You should see following result if the installation is successful.

feature.node.kubernetes.io/pci-10de.present=true

feature.node.kubernetes.io/pci-10de.present=true

## Step 5: Installing GPU Operator

1. Create a yaml file named nvidia\_gpu\_operator\_install.yaml on your local machine.

apiVersion: v1

kind: Namespace

metadata:

name: nvidia-gpu-operator

---

apiVersion: operators.coreos.com/v1

kind: OperatorGroup

metadata:

name: nvidia-gpu-operator-group

namespace: nvidia-gpu-operator

spec:

targetNamespaces:

- nvidia-gpu-operator

---

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: gpu-operator-certified

namespace: nvidia-gpu-operator

spec:

channel: "v25.3"

installPlanApproval: Manual

name: gpu-operator-certified

source: certified-operators

sourceNamespace: openshift-marketplace

startingCSV: "gpu-operator-certified.v25.3.4"

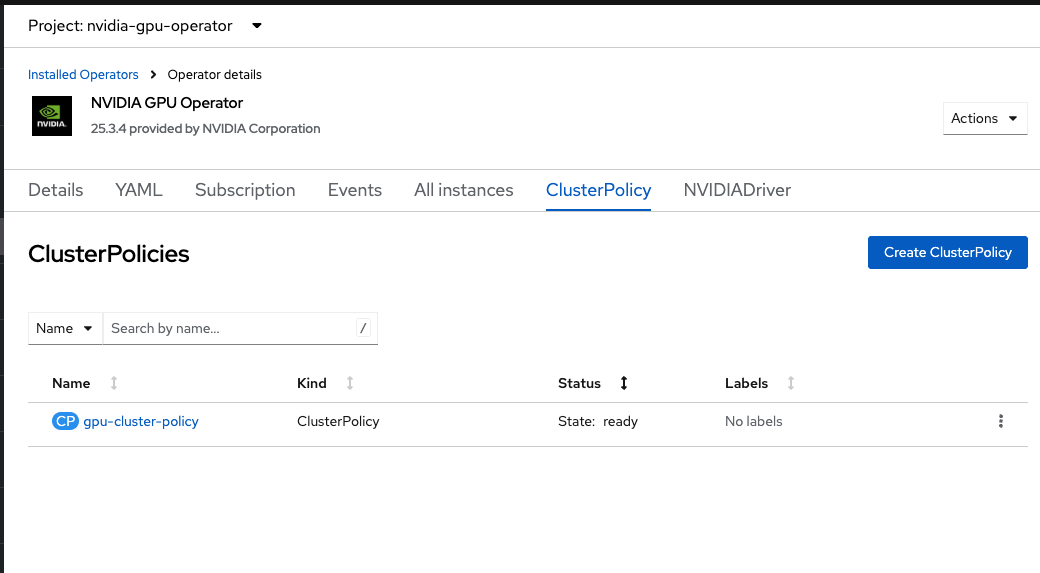
1. Deploy the yaml file.

oc apply -f nvidia\_gpu\_operator\_install.yaml

1. Create the **ClusterPolicy** Instance

* Navigate to NVIDIA GPU Operator.
* Click on Cluster Policy.
* Click on **Create ClusterPolicy**.
* Take all the defaults and click on **Create**.

It will take about 10-20 minutes for the clusterpolicy to be created.



1. Run the following command to view and verify these new pods and daemonsets.

oc get pods,daemonset -n nvidia-gpu-operator

you should see following result if the installation is successful.

NAME READY STATUS RESTARTS AGE

pod/gpu-feature-discovery-c2rfm 1/1 Running 0 6m28s

pod/gpu-operator-84b7f5bcb9-vqds7 1/1 Running 0 39m

pod/nvidia-container-toolkit-daemonset-pgcrf 1/1 Running 0 6m28s

pod/nvidia-cuda-validator-p8gv2 0/1 Completed 0 99s

pod/nvidia-dcgm-exporter-kv6k8 1/1 Running 0 6m28s

pod/nvidia-dcgm-tpsps 1/1 Running 0 6m28s

pod/nvidia-device-plugin-daemonset-gbn55 1/1 Running 0 6m28s

pod/nvidia-device-plugin-validator-z7ltr 0/1 Completed 0 82s

pod/nvidia-driver-daemonset-410.84.202203290245-0-xxgdv 2/2 Running 0 6m28s

pod/nvidia-node-status-exporter-snmsm 1/1 Running 0 6m28s

pod/nvidia-operator-validator-6pfk6 1/1 Running 0 6m28s

NAME DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE SELECTOR AGE

daemonset.apps/gpu-feature-discovery 1 1 1 1 1 nvidia.com/gpu.deploy.gpu-feature-discovery=true 6m28s

daemonset.apps/nvidia-container-toolkit-daemonset 1 1 1 1 1 nvidia.com/gpu.deploy.container-toolkit=true 6m28s

daemonset.apps/nvidia-dcgm 1 1 1 1 1 nvidia.com/gpu.deploy.dcgm=true 6m28s

daemonset.apps/nvidia-dcgm-exporter 1 1 1 1 1 nvidia.com/gpu.deploy.dcgm-exporter=true 6m28s

daemonset.apps/nvidia-device-plugin-daemonset 1 1 1 1 1 nvidia.com/gpu.deploy.device-plugin=true 6m28s

daemonset.apps/nvidia-driver-daemonset-410.84.202203290245-0 1 1 1 1 1 feature.node.kubernetes.io/system-os\_release.OSTREE\_VERSION=410.84.202203290245-0,nvidia.com/gpu.deploy.driver=true 6m28s

daemonset.apps/nvidia-mig-manager 0 0 0 0 0 nvidia.com/gpu.deploy.mig-manager=true 6m28s

daemonset.apps/nvidia-node-status-exporter 1 1 1 1 1 nvidia.com/gpu.deploy.node-status-exporter=true 6m29s

daemonset.apps/nvidia-operator-validator 1 1 1 1 1 nvidia.com/gpu.deploy.operator-validator=true 6m28s

## Step 6: Install RH OCP AI

ROKS includes an OpenShift AI Add-on. When installing this add-on, it automatically deploys the Node Feature Discovery (NFD) and NVIDIA Operator. However, these installations can sometimes fail, particularly during cluster policy creation. To avoid such issues, it is recommended to manually install NFD and the NVIDIA GPU Operator (Steps provided above) before creating the OpenShift AI add-on.

1. Install Service Mesh Operator

oc apply -f - <<EOF

apiVersion: operators.coreos.com/v1alpha1

kind: Subscription

metadata:

name: servicemeshoperator

namespace: openshift-operators

spec:

channel: stable

name: servicemeshoperator

source: redhat-operators

sourceNamespace: openshift-marketplace

EOF

1. Install OCP AI add-on for ROKS

* Navigate to **Containers**->**Cluster management**->**Clusters**
* Click on the cluster you created.
* Scroll down to find Red Hat OpenShift AI add-on, click **Install**.

A screenshot of a chat

AI-generated content may be incorrect.

* Disable NFD Operator and Nvidia GPU operator as you already installed.
* Click on **Install**.

A screenshot of a computer

AI-generated content may be incorrect.

1. Check cluster health

* Login to the cluster

eval $OC\_LOGIN

* Run the cluster command health check command.

cpd-cli health cluster

* Run the nodes health check command.

cpd-cli health nodes

* Run the network-performance command.

cpd-cli health network-performance

* Confirm all above commands return results with “Successful” status.

# Install software Hub Software

## Step 1: Review license terms (Optional for POC)

eval $CPDM\_OC\_LOGIN

cpd-cli manage get-license \

--release=$VERSION \

--component=watsonx\_ai \

--license-type=WXAI

## Step 2: Install the required components for an instance of IBM Software Hub.

1. Deploy CPD operators and operands

cpd-cli manage setup-instance \

--release=$VERSION \

--license\_acceptance=true \

--cpd\_operator\_ns=$PROJECT\_CPD\_INST\_OPERATORS \

--cpd\_instance\_ns=$PROJECT\_CPD\_INST\_OPERANDS \

--block\_storage\_class=$STG\_CLASS\_BLOCK \

--file\_storage\_class=$STG\_CLASS\_FILE \

--run\_storage\_tests=true

The installation will take about **20-30** minutes. In addition, many of the deployed pods by this installation uses service accounts for pull secrets. The installation will have many image-pull errors without patching the service accounts.

1. Patch service accounts using the following command in another terminal window.

for sa in $(oc get sa -n cpd-operand -o jsonpath='{.items[\*].metadata.name}'); do  
oc patch sa $sa -n cpd-operand -p '{"imagePullSecrets":[{"name":"ibm-entitlement-key"}]}' --type=merge  
done

## Step 3: Confirm Status

1. Confirm that the status of the operands is completed.

cpd-cli manage get-cr-status \

--cpd\_instance\_ns=$PROJECT\_CPD\_INST\_OPERANDS

1. Check the health of the resources in the operators project.

cpd-cli health operators \

--operator\_ns=$PROJECT\_CPD\_INST\_OPERATORS \

--control\_plane\_ns=$PROJECT\_CPD\_INST\_OPERANDS

1. Check the health of the resources in the operands project.

cpd-cli health operands \

--control\_plane\_ns=$PROJECT\_CPD\_INST\_OPERANDS

## Step 4: Get the URL and default credentials of the web client.

cpd-cli manage get-cpd-instance-details \

--cpd\_instance\_ns=$PROJECT\_CPD\_INST\_OPERANDS \

--get\_admin\_initial\_credentials=true

expected output:

CPD Url: cpd-cpd-instance.apps.66f2b2761f6d05e1ff70b8cb.ocp.techzone.ibm.com

CPD Username: cpadmin

CPD Password: TuvXdQD43vrPjxM8f60gX7RPINR7joXI

# Deploy watsonx.ai

## Step1: Create the custom resource for IBM watsonx.ai

1. Create installation parameter file

* Create a file called install-options.yaml file in the cpd-cli work directory. (For example: cpd-cli-workspace/olm-utils-workspace/work).

vi $CPD\_CLI\_MANAGE\_WORKSPACE/work/install-options.yaml

1. Copy the following watsonx.ai parameters content to the file.

########################################################################

# watsonx.ai parameters

########################################################################

custom\_spec:

watsonx\_ai:

tuning\_disabled: false

lite\_install: false

## Step 2: create the required OLM objects for watsonx.ai

1. Run the following command to create the objects in the operators project for the instance.

cpd-cli manage apply-olm \

--release=$VERSION \

--cpd\_operator\_ns=$PROJECT\_CPD\_INST\_OPERATORS \

--components=watsonx\_ai

1. Verify result.

Wait for the cpd-cli to return the following message before you proceed to the next step.

[SUCCESS]... The apply-olm command ran successfully

## Step 3: Create the custom resource for IBM watsonx.ai

1. Run the following command.

cpd-cli manage apply-cr \

--components=watsonx\_ai \

--release=$VERSION \

--cpd\_instance\_ns=$PROJECT\_CPD\_INST\_OPERANDS \

--block\_storage\_class=$STG\_CLASS\_BLOCK \

--file\_storage\_class=$STG\_CLASS\_FILE \

--param-file=$CPD\_CLI\_MANAGE\_WORKSPACE/work/install-options.yaml \

--license\_acceptance=true

1. Validating the installation

IBM watsonx.ai is installed when the apply-cr command returns:

[SUCCESS]... The apply-cr command ran successfully

1. Confirm that the custom resource status is completed

cpd-cli manage get-cr-status \

--cpd\_instance\_ns=$PROJECT\_CPD\_INST\_OPERANDS \

--components=Watson\_ai

expected output:

Component CR-kind CR-name Namespace Status Version Creationtimestamp Reconciled-version Operator-info

----------- --------- ------------ ------------ --------- --------- -------------------- -------------------- ---------------

watsonx\_ai Watsonxai watsonxai-cr cpd-instance Completed 9.1.0 2024-09-24T22:29:40Z 9.1.0 34

# 

# Installing Models with Default Configuration

Once you decided the LLM to be deploy, execute the following command. You can deploy more than one model at a time.

## Step 1: Determine models to be deployed and their IDs

The LLM mode information can be found [Here](https://www.ibm.com/docs/en/software-hub/5.2.x?topic=install-foundation-models).

## Step 2: Installing models with the default configuration

1. Run the following command to modify the custom resource. Replace the model-ids with the actual model id.

oc patch watsonxaiifm watsonxaiifm-cr \

--namespace=$PROJECT\_CPD\_INST\_OPERANDS \

--type=merge \

--patch='{"spec":{"install\_model\_list": ["<model-id1>","<model-id2>"]}}'

1. Confirm that the spec section of the watsonxaiifm custom resource is updated by running the following command.

oc describe watsonxaiifm watsonxaiifm-cr -n $PROJECT\_CPD\_INST\_OPERANDS

Wait for the operator to finish reconciling the changes and show the Completed status. You can use the following command to check the status of the service:

oc get watsonxaiifm -n $PROJECT\_CPD\_INST\_OPERANDS

NAME VERSION RECONCILED STATUS PERCENT AGE

watsonxaiifm-cr 11.1.0 11.1.0 Completed 100% 110m

1. Check if changes were applied successfully

oc get po -n ${PROJECT\_CPD\_INST\_OPERANDS} | grep predictor