# Deploying a instance on a cluster

# Kubernetes setup

minikube start --memory 4000 --cpus 3 --disk-size=40GB

# Kube Virt Setup

First, we select the latest version of KubeVirt and set the environment variable KUBEVIRT\_VERSION for future commands.

KUBEVIRT\_VERSION=$(curl -s https://api.github.com/repos/kubevirt/kubevirt/releases/latest | awk -F '[ \t":]+' '/tag\_name/ {print $3}')

# Install KubeVirt with latest version

This command creates a “kubevirt” namespace and installs Custom Resource Definitions for KubeVirt. It also sets up an operator to initiate the KubeVirt installation once a configuration resource is detected.

kubectl create -f https://github.com/kubevirt/kubevirt/releases/download/${KUBEVIRT\_VERSION}/kubevirt-operator.yaml

Now Here We will be adding the Kubevirt Custom Resource (named kubevirt). This will prompt the KubeVirt operator to install the remaining components of KubeVirt.

kubectl create -f https://github.com/kubevirt/kubevirt/releases/download/${KUBEVIRT\_VERSION}/kubevirt-cr.yaml

Working with minikube, direct access to local virtualization hardware might not be available. we may need to enable emulation mode.

kubectl -n kubevirt patch kubevirt kubevirt --type=merge --patch '{"spec":{"configuration":{"developerConfiguration":{"useEmulation":true}}}}'

# Install the KubeVirt client, virtctl.

Until we complete the setup, we can download the virtctl command line client for KubeVirt.

curl -Lo virtctl https://github.com/kubevirt/kubevirt/releases/download/${KUBEVIRT\_VERSION}/virtctl-${KUBEVIRT\_VERSION}-linux-amd64

To ensure the downloaded binary is executable, execute the following command:

chmod +x virtctl

Put the virtctl binary in a directory included in your shell’s PATH

mv virtctl $HOME/.local/bin  
mv virtctl /usr/local/bin

*Wait for KubeVirt to finish deploying.* We can check its deployment status by examining the phase of the Kubevirt Custom Resource (CR). Use the following command:

kubectl -n kubevirt get kubevirt

Once KubeVirt fully deploys, it will show:

NAME AGE PHASE  
kubevirt 3m Deployed

To get more details and pods deployed we can use

kubectl get pods -n kubevirt

Once the deployment completes, confirm all necessary Pods are running without issues.

# Launch a Instance

To create a virtual machine, install the VM manifest using the following command:

kubectl apply -f https://kubevirt.io/labs/manifests/vm.yaml

Check the status of the virtual machine we just created:

kubectl get vm

check the status to be RUNNING if Failed we can see the detailed status using

kubectl describe vm VM\_NAME

If the VM is currently in a stopped state. We can use virtctl to change its state to running:

virtctl start testvm

We will see a message that the VM was scheduled to start.

A VirtualMachine manages a VirtualMachineInstance, much like how a Deployment manages Pods. So, when we create a VirtualMachine, it generates a VirtualMachineInstance, which in turn creates a Pod. We can see all these entities with just one ‘kubectl get’ command.

# *Connect to VM*

Once the virtual machine is running, we can connect to its serial console using virtctl: virtctl console testvm

To Exit press CTRL and “]” keys together

# To stop and delete a virtual machine:

Stop the virtual machine using virtctl from outside the VM:

virtctl stop testvm

To delete the virtual machine

kubectl delete virtualmachine testvm

# Persistent Storage for Virtual Machines:

For persistent storage in virtual machines, the KubeVirt project offers a solution through the Containerized Data Importer (CDI) add-on. CDI facilitates the import of virtual machine disk images into PersistentVolumeClaims (PVCs), providing a declarative approach to managing persistent storage.

# Upload with virtctl:

If we have a local copy of the image we want to use, virtctl provides an image-upload command.

virtctl image-upload dv <datavolume\_name> --size=<datavolume\_size> --image-path=</path/to/image> \

# Create a DataVolume:

CDI introduces a CustomResourceDefinition (CRD) called DataVolume (DV), which abstracts the creation and population of disk data onto PVCs in a Kubernetes cluster. Once a DV is created, CDI handles the process of copying a disk image from the specified source into a PVC. Various source types are supported, including external targets like images served over HTTP, internal sources like other PVCs, or container disks in a registry. Below is an example DataVolume that imports a CirrOS image into a PVC, enabling the underlying VM to have persistent storage across reboots.

# Install CDI

Get the Latest Version

export VERSION=$(curl -Ls https://github.com/kubevirt/containerized-data-importer/releases/latest | grep -m 1 -o "v[0-9]\.[0-9]\*\.[0-9]\*")

Install cdi

kubectl create -f https://github.com/kubevirt/containerized-data-importer/releases/download/$VERSION/cdi-operator.yaml

To trigger the deployment of CDI, create a CDI Custom Resource (CR). This will prompt the operator to deploy CDI.

kubectl create -f https://github.com/kubevirt/containerized-data-importer/releases/download/$VERSION/cdi-cr.yaml

Wait until it gets Deployed

NAME AGE PHASE cdi 40h Deployed

Once Deployed we are Ready to Work Further # Create a Data Volume from the required image:

ubuntu-dv.yaml

---   
apiVersion: cdi.kubevirt.io/v1beta1  
kind: DataVolume  
metadata:  
 name: ubuntu  
spec:  
 source:  
 http:  
 url: https://cloud-images.ubuntu.com/focal/current/focal-server-cloudimg-amd64.img  
 pvc:  
 accessModes:  
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 3Gi

Get it Executed using the following command We can change the url and storage according to our requirement

kubectl create -f ubuntu-dv.yaml

To check the DV status:

kubectl get dv

# Create a Instance from the Data Volume

Relace HASHED\_PASSWORD, SSH\_PUB\_KEY\_HERE with values ubuntu-vm.yaml

---  
apiVersion: kubevirt.io/v1  
kind: VirtualMachine  
metadata:  
 labels:  
 kubevirt.io/os: linux  
 name: ubuntu-vm  
spec:  
 running: true  
 template:  
 metadata:  
 creationTimestamp: null  
 labels:  
 kubevirt.io/domain: ubuntu-vm  
 spec:  
 domain:  
 cpu:  
 cores: 2  
 devices:  
 disks:  
 - disk:  
 bus: virtio  
 name: disk0  
 - cdrom:  
 bus: sata  
 readonly: true  
 name: cloudinitdisk  
 resources:  
 requests:  
 memory: 512M  
 volumes:  
 - name: disk0  
 persistentVolumeClaim:  
 claimName: ubuntu  
 - cloudInitNoCloud:  
 userData: |  
 #cloud-config  
 users:   
 - default  
 - name: yash  
 passwd: HASHED\_PASSWORD  
 lock\_passwd: false  
 shell: /bin/bash  
 ssh\_pwauth: True  
 chpasswd: { expire: False}  
 sudo: ALL=(ALL) NOPASSWD:ALL  
 groups: users, admin  
 ssh\_authorized\_keys:  
 - ssh-rsa SSH\_PUB\_KEY\_HERE  
 name: cloudinitdisk

Get it Executed

kubectl create -f ubuntu-vm.yaml

We can get the created Resources:

kubectl get dv,pvc,vm

Once the status of vm becomes “RUNNING” and Ready is “True”

We could use the command to access the instance

virtctl console VM-NAME

If needs to connect through VNC

virtctl vnc VM-NAME