Using KubeVirt on SUSE Linux Enterprise

Jim Fehlig and Vasily Ulyanov

KubeVirt is a virtual machine management add-on for Kubernetes. KubeVirt extends Kubernetes by adding additional virtualization resource types through Kubernetes' Custom Resource Definitions (CRD) API. Along with the Custom Resources, KubeVirt includes controllers and agents that provide virtual machine management capabilities on the cluster. By using this mechanism, the Kubernetes API can be used to manage virtual machine resources similar to other Kubernetes resources.

Publication Date: May 15, 2025

Contents

1 KubeVirt components 3

2 Installing KubeVirt on Kubernetes 4

3 Updating the KubeVirt deployment 5

4 Deleting KubeVirt from a cluster 5

5 Containerized Data Importer 6

6 Running virtual machines 6

7 Live migration 8

8 Volume hotplugging 10

9 Running Windows VMs with VMDP ISO 10

10 Supported features 11

11 Debugging 13

1 KubeVirt components

KubeVirt consists of two RPM-based packages and six container images that provide the Kubernetes virtual machine management extension. The RPM packages include `kubevirt-virtctl` and `kubevirt-manifests`. The container images include `virt-api`, `virt-controller`, `virt-handler`, `virt-launcher`, and `virt-operator`, `libguestfs-tools`.

`kubevirt-virtctl` can be installed on any machine with administrator access to the cluster. It contains the `virtctl` tool, which provides syntactic sugar on top of the `kubectl` tool for virtual machine resources. Although the `kubectl` tool can be used to manage virtual machines, it is a bit awkward since, unlike standard Kubernetes resources, virtual machines maintain state. Migration is also unique to virtual machines. If a standard Kubernetes resource needs to be evacuated from a cluster node, it is destroyed and started again on an alternate node. Since virtual machines are stateful, they cannot be destroyed and must be live-migrated away if a node is under evacuation. The `virtctl` tool abstracts the complexity of managing virtual machines with `kubectl`. It can be used to stop, start, pause, unpause and migrate virtual machines. `virtclt` also provides access to the virtual machine's serial console and graphics server.

`kubevirt-manifests` contains the manifests, or recipes, for installing KubeVirt. The most interesting files are `/usr/share/kube-virt/manifests/release/kubevirt-cr.yaml` and `/usr/share/kube-virt/manifests/release/kubevirt-operator.yaml`. `kubevirt-cr.yaml` contains the KubeVirt Custom Resource definition that represents the KubeVirt service. `kubevirt-operator.yaml` is the recipe for deploying the KubeVirt operator, which deploys the KubeVirt service to the cluster and manages its' lifecycle.

`virt-api` is a cluster component that provides the Kubernetes API extension for virtual machine resources. Like `virt-api`, `virt-controller` is a cluster component that watches for new objects created via `virt-api`, or updates to existing objects, and takes action to ensure the object state matches the requested state. `virt-handler` is a DaemonSet and a node component that has the job of keeping the cluster-level virtual machine object in sync with the `libvirtd` domain running in `virt-launcher`. `virt-handler` can also perform node-centric operations like configuring networking and/or storage on the node per the virtual machine specification. `virt-launcher` is also a node component and has the job of running `libvirt` plus `qemu` to provide the virtual machine environment. `virt-launcher` is a lowly pod resource. `libguestfs-tools` is a component providing a set of utilities for accessing and modifying VM disk images.

`virt-operator` implements the Kubernetes operator pattern. Operators encode the human knowledge required to deploy, run and maintain an application. Operators are a Kubernetes Deployment resource type and are often used to manage the custom resources and custom controllers that together provide a more complex Kubernetes application such as KubeVirt.

2 Installing KubeVirt on Kubernetes

KubeVirt can be installed on a Kubernetes cluster by installing the `kubevirt-manifests` package on an admin node, applying the `virt-operator` manifest, and creating the KubeVirt custom resource. For example, on a cluster admin node execute the following:

```
> sudo zypper install kubevirt-manifests
> kubectl apply -f /usr/share/kube-virt/manifests/release/kubevirt-operator.yaml
> kubectl apply -f /usr/share/kube-virt/manifests/release/kubevirt-cr.yaml
```

After creating the KubeVirt custom resource, `virt-operator` deploys the remaining KubeVirt components. Progress can be monitored by viewing the status of the resources in the `kubevirt` namespace:

```
> kubectl get all -n kubevirt
```

The cluster is ready to deploy virtual machines once `virt-api`, `virt-controller`, and `virt-handler` are READY with STATUS “Running”.

Alternatively it is possible to wait until KubeVirt custom resource becomes available:

```
> kubectl -n kubevirt wait kv kubevirt --for condition=Available
```

Some KubeVirt functionality is disabled by default and must be enabled via feature gates. For example, live migration and the use of HostDisk for virtual machine disk images are disabled. Enabling KubeVirt feature gates can be done by altering an existing KubeVirt custom resource and specifying the list of features to enable. For example, you can enable live migration and the use of HostDisks:

```
> kubectl edit kubevirt kubevirt -n kubevirt
    ...
    spec:
      configuration:
        developerConfiguration:
          featureGates:
```

```
            - HostDisk
            - LiveMigration
```

Note

The names of feature gates are case-sensitive.

3 Updating the KubeVirt deployment

Updating KubeVirt is similar to the initial installation. The updated operator manifest from the `kubevirt-manifests` package is applied to the cluster.

```
> sudo zypper update kubevirt-manifests
> kubectl apply -f /usr/share/kube-virt/manifests/release/kubevirt-operator.yaml
```

4 Deleting KubeVirt from a cluster

KubeVirt can be deleted from a cluster by deleting the custom resource and operator:

```
> kubectl delete -n kubevirt kubevirt kubevirt # or alternatively: kubectl delete -f /
usr/share/kube-virt/manifests/release/kubevirt-cr.yaml
> kubectl delete -f /usr/share/kube-virt/manifests/release/kubevirt-operator.yaml
```

Note

It is important to delete the custom resource first otherwise it gets stuck in the `Terminating` state. To fix that the resource finalizer needs to be manually deleted:

```
> kubectl -n kubevirt patch kv kubevirt --type=json -p '[{ "op": "remove", "path":
 "/metadata/finalizers" }]'
```

After deleting the resources from Kubernetes cluster the installed KubeVirt RPMs can be removed from the system:

```
> sudo zypper rm kubevirt-manifests kubevirt-virtctl
```

5 Containerized Data Importer

Containerized Data Importer (CDI) is an add-on for Kubernetes focused on persistent storage management. It is primarily used for building and importing Virtual Machine Disks for KubeVirt.

5.1 Installing CDI

CDI can be installed on a Kubernetes cluster in a way similar to KubeVirt by installing the RPMs and applying the operator and custom resource manifests using kubectl:

```
> sudo zypper in containerized-data-importer-manifests
> kubectl apply -f /usr/share/cdi/manifests/release/cdi-operator.yaml
> kubectl apply -f /usr/share/cdi/manifests/release/cdi-cr.yaml
```

5.2 Updating and deleting CDI:

To update CDI:

```
> sudo zypper update containerized-data-importer-manifests
> kubectl apply -f /usr/share/cdi/manifests/release/cdi-operator.yaml
```

To delete CDI:

```
> kubectl delete -f /usr/share/cdi/manifests/release/cdi-cr.yaml
> kubectl delete -f /usr/share/cdi/manifests/release/cdi-operator.yaml
> sudo zypper rm containerized-data-importer-manifests
```

6 Running virtual machines

Two of the most interesting custom resources provided by KubeVirt are *VirtualMachine* (VM) and *VirtualMachineInstance* (VMI). As the names imply, a VMI is a running instance of a VM. The lifecycle of a VMI can be managed independently from a VM, but long-lived, stateful virtual machines are managed as a VM. The VM is deployed to the cluster in a shutoff state, then activated by changing the desired state to running. Changing a VM resource state can be done with the standard Kubernetes client tool `kubectl` or with the client `virtctl` provided by KubeVirt.

The VM and VMI custom resources make up part of the KubeVirt API. To create a virtual machine, a VM or VMI manifest must be created that adheres to the API. The API supports setting a wide variety of the common virtual machine attributes, for example, model of vCPU, number

of vCPUs, amount of memory, disks, network ports, etc. Below is a simple example of a VMI manifest for a virtual machine with one Nehalem CPU, 2G of memory, one disk, and one network interface:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-host-disk
  name: sles15sp2
spec:
  domain:
    cpu:
      model: Nehalem-IBRS
    devices:
      disks:
      - disk:
          bus: virtio
        name: host-disk
      interfaces:
        - name: green
          masquerade: {}
          ports:
            - port: 80
    machine:
      type: ""
    resources:
      requests:
        memory: 2048M
  terminationGracePeriodSeconds: 0
  networks:
  - name: green
    pod: {}
  volumes:
  - hostDisk:
      path: /hostDisks/sles15sp2/disk.raw
      type: Disk
      shared: true
    name: host-disk
```

Applying this VMI manifest to the cluster creates a virt-launcher container running `libvirt` and `qemu`, providing the familiar KVM virtual machine environment.

```
> kubectl apply -f sles15sp2vmi.yaml
> kubectl get vmis
```

Similar to other Kubernetes resources, VMs and VMIs can be managed with the `kubectl` client tool. Any `kubectl` operation that works with resource types works with the KubeVirt custom resources, for example, describe, delete, get, log, patch, etc. VM resources are a bit more awkward to manage with `kubectl`. Since a VM resource can be in a shutoff state, turning it on requires patching the manifest to change the desired state to running. Find an example below:

```
> kubectl patch vm sles15sp2 --type merge -p '{"spec":{"running":true}}'
```

The `virtctl` tool included in the `kubevirt-virtclt` package provides syntactic sugar on top of `kubectl` for VM and VMI resources, allowing them to be stopped, started, paused, unpaused and migrated. `virtctl` also provides access to the virtual machine's serial console and graphics server. Find an example below:

```
> virtctl start VM
> virtctl console VMI
> virtctl stop VM
> virtctl pause VM|VMI
> virtctl unpause VM|VMI
> virtctl vnc VMI
> virtctl migrate VM
```

7 Live migration

KubeVirt supports live migration of VMs. Though this functionality must first be activated by adding `LiveMigration` to the list of feature gates in the KubeVirt custom resource.

```
> kubectl edit kubevirt kubevirt -n kubevirt
```

```
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - LiveMigration
```

7.1 Prerequisites

- All the Persistent Volume Claims (PVCs) used by a VM must have `ReadWriteMany` (RWX) access mode.
- VM pod network binding must be of type `masquerade`:

```
spec:
  domain:
    devices:
      interfaces:
        - name: green
          masquerade: {}
```

Whether live migration is possible or not can be checked via the `VMI.status.conditions` field of a running VM spec:

```
> kubectl describe vmi sles15sp2
```

```
Status:
  Conditions:
    Status: True
    Type: LiveMigratable
  Migration Method: BlockMigration
```

7.2 Initiating live migration

Live migration of a VMI can be initiated by applying the following yaml file:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
spec:
  vmiName: sles15sp2
```

```
> kubectl apply -f migration-job.yaml
```

Alternatively it is possible to migrate a VM using `virtctl` tool:

```
> virtctl migrate VM
```

7.3 Cancelling live migration

Live migration can be canceled by deleting the existing migration object:

```
> kubectl delete VirtualMachineInstanceMigration migration-job
```

8 Volume hotplugging

KubeVirt allows hotplugging additional storage into a running VM. Both block and file system volume types are supported. The hotplug volumes feature can be activated via the `HotplugVolumes` feature gate:

```
> kubectl edit kubevirt kubevirt -n kubevirt
```

```
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - HotplugVolumes
```

Assuming that `hp-volume` is an existing DataVolume or PVC, `virtctl` can be used to operate with the volume on a runnig VM:

```
> virtctl addvolume sles15sp2 --volume-name=hp-volume
> virtctl removevolume sles15sp2 --volume-name=hp-volume
```

9 Running Windows VMs with VMDP ISO

The VMDP ISO is provided in the form of a container image which can be consumed by KubeVirt. To run a Windows VM with VMDP ISO attached, the corresponding `containerDisk` needs to be added to the VM definition:

```
spec:
  domain:
    devices:
      disks:
        - name: vmdp
          cdrom:
            bus: sata
```

```
volumes:
  - containerDisk:
      image: registry.suse.com/suse/vmdp/vmdp:latest
    name: vmdp
```

Note

The sequence in which the disks are defined affects the boot order. It is possible to specify the `bootOrder` explicitly or otherwise sort the disk items as needed.

10 Supported features

- Guest Agent Information
- Live migration
- Hotplug volumes
- VMI Dedicated CPU resource

10.1 VMI virtual hardware

- machine type
- BIOS/UEFI/SMBIOS
- cpu
- clock
- RNG
- CPU/Memory limits and requirements
- tablet input
- hugepage

10.2 VMI disks and volumes

Disk types:

- lun
- disk
- cdrom

Volume sources:

- cloudInitNoCloud
- cloudInitConfigDrive
- persistentVolumeClaim
- dataVolume
- ephemeral
- containerDisk
- emptyDisk
- hostDisk
- configMap
- secret
- serviceAccount
- downwardMetrics

High performance features:

- IO threads
- Virtio Block Multi-Queue
- Disk cache

10.3 VMI interfaces and networks

Network (back-end) types:

- pod
- multus

Interface (front-end) types:

- bridge
- masquerade

11 Debugging

If issues are encountered the following debug resources are available to help identify the problem.

The status of all KubeVirt resources can be examined with the **`kubectl get`** command:

```
> kubectl get all -n kubevirt
```

Resources with failed status can be further queried by examining their definition and expanded status information.

```
> kubectl describe deployment virt-operator
> kubectl get deployment virt-operator -o yaml -n kubevirt
> kubectl describe pod virt-handler-xbjkg -n kubevirt
> kubectl get pod virt-handler-xbjkg -o yaml -n kubevirt
```

Logs from the problematic KubeVirt pod can contain a wealth of information since `stderr` and service logging from within the pod is generally available via the Kubernetes log service:

```
> kubectl logs virt-operator-558c57bc4-mg68w -n kubevirt
  > kubectl logs virt-handler-xbjkg -n kubevirt
```

If the underlying pod is running but there are problems with the service running in it, a shell can be accessed to inspect the pod environment and poke at its service:

```
> kubectl -n kubevirt exec -it virt-handler-xbjkg -- /bin/bash
```