Virtualization

Overview

Introduction

Container-Orchestrated Virtual Machine Solution

Features

Product Features

How Console Features Map to API Objects

Architecture and Permissions

Constraints and Limitations

Concepts and Glossary

Concepts and Glossary

How it fits together

Compute objects

Images and disks

Networking

Migration and availability

Backup and recovery

Where to start

Install

Install

Prerequisites

Planning and Requirements

Procedure

Resource Quota Explanation

Images

Introduction

Advantages

Uploading Images from Outside the Cluste

Guides

How To

Permissions

Virtual Machine

Introduction

Bootable Volumes

Notes

Guides

How To

Troublesho

Create a virtual machine from a bootable volume

Update or delete a bootable volume

Network

Introduction

Advantages

Guides

How To

Storage

Introduction

Advantages

Guides

Backup and Recovery

Introduction

Application Scenarios

Usage Limitations

Guides

Overview

Introduction

Introduction

For enterprises using a virtual machine-based architecture, transitioning to a Kubernetes and container-based architecture inevitably requires application modernization. However, due to constraints such as the need for continuous business uptime or the difficulty in changing development habits, enterprises often cannot completely disengage from virtualization architecture in a short period.

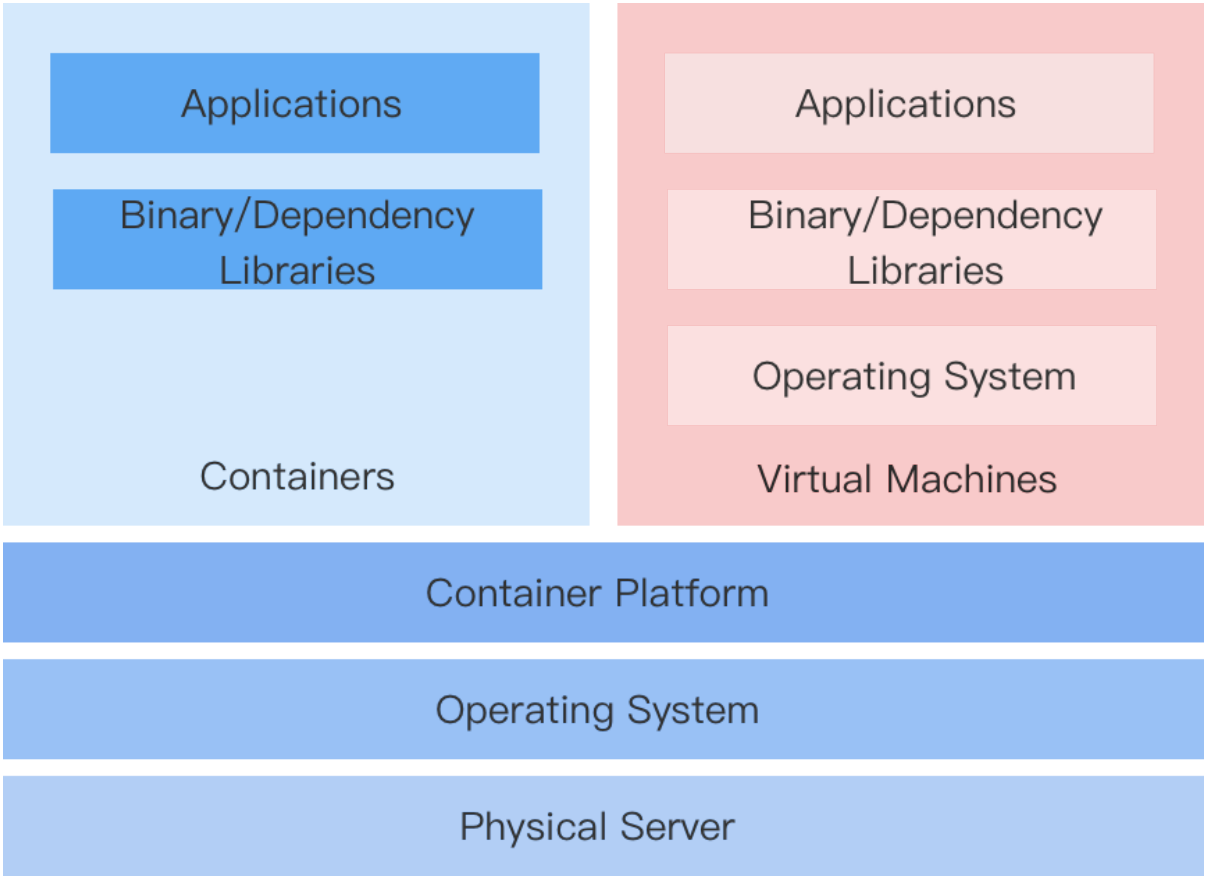
Therefore, a solution that can uniformly configure, manage, and control container resources and virtual machine resources on the same platform becomes particularly important.

TOC

Container-Orchestrated Virtual Machine Solution

This platform implements a virtual machine (VMI, VirtualMachineInstance) solution based on the open-source component KubeVirt, allowing for easier and faster creation of container-

orchestrated virtual machines and running virtualized applications.



Features

Rapid Transformation

There is no need to rewrite applications or modify images. Simply package the existing application into a qcow2 or raw format virtual machine image, and create a virtual machine using that image on the platform, allowing the application to be deployed to the container platform.

Maintain Behavioral Habits

Containerized virtual machines can be managed using a similar approach to traditional virtual machines, without needing to focus on the underlying container implementation, including virtual machine lifecycle management, disks and networks, and snapshot management.

Coexistence of Virtualization and Containerization

- The unified platform supports managing virtualized services while also enabling Kubernetes-based container scheduling and management.

- On the basis of continuing to use virtual machine workloads, it allows for a gradual modernization of containerized applications.
- The development of new containerized applications that need to interact with virtualized applications remains unaffected.

Product Features

- **Virtual Machine**: Supports creating virtual machines with images allocated by administrators and managing them, including starting and stopping virtual machines, managing snapshots, remote login to virtual machines, and modifying virtual machine configurations.
- **Virtual Disk**: Supports viewing and managing disk information created in the current project, including creating disks, viewing disk names, storage classes, capacities, and associated virtual machines.
- **Virtual Machine Snapshots**: Supports viewing details such as the status of virtual machine snapshots, the associated virtual machine, and the most recent rollback time.
- **Virtual Machine Images**: Supports viewing virtual machine image information under the current project, including image provision method and operating system.
- **Key Pairs**: Supports viewing and managing key pairs created in the current project, including creating key pairs and viewing the list of associated virtual machines.
- **Bootable Volumes**: Manage golden-image boot sources across namespaces, with four import sources (HTTP/HTTPS, container registry, clone PVC, clone snapshot).
- **Live Migration**: Migrate a running virtual machine between nodes without downtime, governed by cluster-wide migration limits and per-namespace migration policies.
- **Storage Migration**: Move a running virtual machine's disks between storage classes online.
- **More lifecycle actions**: Clone a whole virtual machine, pause and resume it, and open a serial console (text TTY) or VNC console.

How Console Features Map to API Objects

Console feature	Kubernetes resource
Virtual machine	`VirtualMachine` (`kubevirt.io/v1`)
Virtual disk	`DataVolume` / PVC (`cdi.kubevirt.io/v1beta1`)
Virtual machine snapshot	`VirtualMachineSnapshot` (`snapshot.kubevirt.io/v1beta1`)
Bootable volume	`DataSource` (`cdi.kubevirt.io/v1beta1`)
Clone	`VirtualMachineClone` (`clone.kubevirt.io/v1beta1`)
Live migration	`VirtualMachineInstanceMigration` (`kubevirt.io/v1`)
Storage migration	`VirtualMachineStorageMigrationPlan` + `VirtualMachineStorageMigration` (`migrations.kubevirt.io/v1alpha1`)
Key pair	`Secret` (type `kubernetes.io/ssh-public-key`)
Pause / Resume, Serial / VNC console	virtual machine instance subresources (`subresources.kubevirt.io`)

Architecture and Permissions

The platform's virtualization is built on KubeVirt and CDI, managed by the HyperConverged (HCO) operator:

- **HyperConverged (HCO)** is the single source of truth — it deploys and continuously reconciles KubeVirt and CDI. Cluster-level settings (such as live-migration limits) must be set on the `HyperConverged` resource, not directly on `KubeVirt` .
- **CDI (Containerized Data Importer)** imports and clones disk images into PVCs — the engine behind bootable volumes and virtual machine disks.
- **virt-controller** schedules virtual machines, **virt-handler** runs on each node, and **virt-launcher** wraps each running virtual machine in a pod.

To delegate access, bind the KubeVirt cluster roles in a namespace:

Cluster role	Grants
`kubevirt.io:view`	Read-only access to virtual machines and instances
`kubevirt.io:edit`	Create / update / delete virtual machines and open consoles
`kubevirt.io:admin`	Full virtual machine management, including subresources
`kubevirt.io:migrate`	Trigger live migrations

CDI provides matching `cdi.kubevirt.io:view` / `:edit` / `:admin` roles for managing DataVolumes and DataSources.

Constraints and Limitations

It must be implemented based on a physical machine cluster, and KubeVirt components must be deployed within the cluster with virtualization enabled. The platform provides the capability to deploy KubeVirt components via Operator and an interface to enable virtualization, with all related configurations completed by the platform administrator.

Concepts and Glossary

This page is for administrators who are fluent in Kubernetes but new to virtualization. The rest of the manual assumes the terms below; each is defined once here, with the closest Kubernetes analogy, so the how-to pages stay short.

TOC

How it fits together

Virtualization on this platform is KubeVirt and CDI, deployed and reconciled by one operator:

- **HyperConverged (HCO)** — the `HyperConverged` custom resource is the single source of truth. The operator installs and continuously reconciles KubeVirt and CDI from it, so cluster-wide settings (for example live-migration limits) are set on `HyperConverged`, **not** directly

on the `KubeVirt` resource. Analogy: a top-level operator CR, like an `Installation` / `Subscription` that owns everything below it.

- **KubeVirt** — runs the virtual machines. Its controllers are **virt-controller** (cluster-level, schedules VMs), **virt-handler** (a DaemonSet, one per node, like the kubelet for VMs), and **virt-launcher** (one pod per running VM that wraps the QEMU process).
- **CDI (Containerized Data Importer)** — imports and clones disk images into PVCs. It is the engine behind bootable volumes and VM disks.

A running virtual machine is therefore an ordinary **Pod** (the virt-launcher pod) on an ordinary node, holding an ordinary **PVC** for its disk and getting an ordinary **Pod IP** — which is why your existing Kubernetes tooling (RBAC, NetworkPolicy, scheduling, monitoring) still applies.

Compute objects

Term	What it is	Kubernetes analogy
VirtualMachine (`kubevirt.io/v1`)	The desired-state definition of a VM (CPU, memory, disks, network, whether it should be running).	A `Deployment` — the spec you create and edit.
VirtualMachineInstance (VMI)	The *running* instance produced when a `VirtualMachine` is started. Lifecycle subresources (console, VNC, pause, migrate) act on the VMI.	A `Pod` — created from the spec, exists only while running.
virt-launcher pod	The pod that wraps one running VM's QEMU process.	The Pod that actually executes the workload.
`spec.running`	Boolean power switch on the `VirtualMachine` (`true` starts it, `false` stops it). This platform uses this field.	`replicas: 1` vs `0`.

Term	What it is	Kubernetes analogy
`spec.runStrategy`	The upstream alternative to `spec.running` (`Always` / `RerunOnFailure` / `Manual` / `Halted`) for finer restart behavior.	A restart policy.
instancetype	A reusable, cluster-wide CPU/memory profile a VM can reference instead of inline resources.	A sizing preset / `LimitRange`-like template.
preference	A reusable profile of OS-specific device/firmware defaults (bus types, clock, features).	A device/firmware "profile" applied to the VM.
QEMU guest agent	An agent installed *inside* the guest OS. Enables IP reporting, graceful shutdown, and filesystem freeze for application-consistent snapshots. Without it, the platform cannot see the guest IP and online snapshots are only crash-consistent.	A sidecar/agent inside the workload that reports health and responds to signals. See Installing Guest Tools.
vTPM	A virtual TPM 2.0 device (`spec.template.spec.domain.devices.tpm`). Required by Windows 11 and BitLocker.	A virtual security device attached to the VM.

Images and disks

A VM boots from a **disk** that is cloned from a reusable **image**. The chain is:

```
remote image (HTTP / registry / S3)  ──CDI import──▶  PVC (a DataVolume)  ──published as──▶  DataSource (a "bootable volume")  ──cloned at VM create──▶  the VM's boot disk
```

Term	What it is	Kubernetes analogy
qcow2 / raw	Disk-image file formats. `qcow2` is sparse/copy-on-write; `raw` is a flat image. This is the *content* you import, not a container image.	A disk image artifact, like an OS cloud image.
containerDisk	A VM disk packaged *inside* a container image and pulled like one. Note: a normal application container image is **not** a containerDisk and will not boot.	An OCI image whose payload is a bootable disk.
CDI / DataVolume	A `DataVolume` (`cdi.kubevirt.io/v1beta1`) declares "import this image into a PVC"; CDI does the import and creates the backing PVC.	A PVC plus an init job that populates it.
DataSource / bootable volume	A `DataSource` (`cdi.kubevirt.io/v1beta1`) points at a golden-image PVC, also called a **Bootable Volume**; creating a VM clones it into the VM's boot disk.	A read-only "golden" PVC used as a template. See Bootable Volumes.
volumeMode: Block vs Filesystem	How the PVC is presented. **Block** performs closer to bare metal and is preferred for VM disks; Filesystem also works.	The PVC `volumeMode` you already know.
StorageProfile / cloneStrategy	A CDI-published object per StorageClass describing supported access/volume modes and how clones are made (`csi-clone`, `snapshot`, or host-assisted `copy`).	Capabilities metadata for a StorageClass. See Managing Virtual Disks.

A fresh cluster has **no** bootable volumes until an administrator imports one (or publishes golden images into `kube-public`). Start at Bootable Volumes.

Networking

By default a VM runs in a virt-launcher **Pod** and uses that Pod's network — so it gets a Pod IP and is subject to the same CNI and `NetworkPolicy` as any Pod. The create form's **Network Mode** maps to a KubeVirt binding:

UI label	YAML binding	Behavior
Bridged	`bridge: {}`	The VM takes over the Pod's network interface and **uses the Pod IP directly**. Required for some protocols; not live-migration friendly by default — live migration over a bridge requires the `kubevirt.io/allow-pod-bridge-network-live-migration` annotation and a supported CNI (see Live Migration).
NAT	`masquerade: {}`	The VM gets a private internal IP and is NAT'd out through the Pod IP. The default; live-migration friendly.

> "**Container group**" in the UI means the Kubernetes **Pod** that backs the VM.

For a routable IP on the physical network (instead of a Pod IP), use a Kube-OVN **Underlay** subnet or an auxiliary NIC. Secondary NICs attach via a `NetworkAttachmentDefinition` (Multus).

Migration and availability

Term	What it is	Kubernet
Live migration	Moving a *running* VM to another node with no downtime, by copying its memory. **Pre-copy** copies memory iteratively while the VM runs; **post-copy** finishes copying after the VM is already on the target.	Reschedu without res process — cannot no Live Migra
VirtualMachineInstanceMigration	The request object that triggers one live migration; deleting it cancels an in-progress migration.	A one-sho moves the
MigrationPolicy	Cluster-scoped overrides (per namespace/VM) for migration tuning. Cluster defaults live on `HyperConverged.spec.liveMigrationConfig`.	A policy C labels.
evictionStrategy	What happens on node drain: `LiveMigrate`, `LiveMigrateIfPossible`, or `None`. A non-migratable VM (RWO storage or device	A `PodDisru` like constr

Term	What it is	Kubernet
	passthrough) with `LiveMigrate` **blocks the drain** — and therefore cluster upgrades.	migrates i blocking. S Availabilit
Live migration requires shared storage	A VM can only live-migrate when all its disk PVCs use **ReadWriteMany (RWX)** access mode (for example CephRBD in Block mode).	RWX vs R the same
Storage migration	Moving a running VM's disks between StorageClasses online.	Re-homin different S without do

Backup and recovery

Term	What it is	Kubernetes analogy
VirtualMachineSnapshot / VirtualMachineRestore	Point-in-time snapshot of a VM's disks (and config) and its restore object. Backed by CSI `VolumeSnapshot`.	A `VolumeSnapshot`, scoped to the whole VM.
crash-consistent vs application-consistent	Without the guest agent a snapshot is *crash-consistent* (like pulling the power). With the guest agent the filesystem is quiesced first, giving an *application-consistent* snapshot.	The fsfreeze distinction familiar from storage backups.

Where to start

A first-time path through the manual:

1. Install — enable virtualization and deploy the HCO operator.
2. Bootable Volumes — import a usable OS image (a fresh cluster has none).
3. Creating Virtual Machines — create your first VM from that image.

4. Serial Console — log in (cloud images have no default password; use the SSH key or password you set at create time).
5. Managing Virtual Disks — attach a data disk.

The overview also has a "Console feature → API object" mapping table, and every how-to page carries a copy-pasteable **Using the API** block.

Install

In order for project personnel to fully utilize virtualization features within the container platform, the platform administrator must perform the following operations to prepare the virtualization environment.

TOC

Prerequisites

- **Download** the **ACP Virtualization with KubeVirt** installation package corresponding to your platform architecture.

- **Upload** the **ACP Virtualization with KubeVirt** installation package using the Upload Packages mechanism.
- When using virtualization features, it is necessary to plan and prepare the network and storage environments in advance.
  Note:
  - If you need to connect to the virtual machine directly via IP, the cluster must use the Kube-OVN Underlay network mode. You can refer to the best practices Preparing Kube-OVN Underlay Physical Network.
  - It is recommended to use TopoLVM with Kubevirt, as it can provide near-hardware-level performance. If performance requirements are not high, Ceph distributed storage can also be used.

Storage Product	Description
TopoLVM	**Advantages**: Relatively lightweight and good performance. **Disadvantages**: Cannot be used across nodes, has low reliability, and cannot provide redundancy.
Ceph Distributed Storage	**Advantages**: Can be used across nodes, highly available, and has redundancy. **Disadvantages**: Redundant disk copies lead to lower utilization; performance is poorer.

  - If TopoLVM is used and multiple disks are configured, please ensure that the remaining storage capacity on the virtualization-enabled nodes can meet the total capacity of the multiple disks; otherwise, the virtual machine creation will fail.
  - If Ceph distributed storage is being used, please ensure that the network where the storage resides and the network where the virtual machines reside can communicate with each other.

Planning and Requirements

Before enabling virtualization, plan the cluster against these requirements:

- **CPU virtualization extensions**: virtualization-enabled nodes must have Intel VT-x or AMD-V enabled in firmware. Mixed CPU models across nodes can prevent live migration.
- **Live migration requires shared storage**: a virtual machine can only be live-migrated when all of its PVCs use the **ReadWriteMany (RWX)** access mode. Plan RWX-capable storage (for example Ceph RBD in Block mode) for workloads that must migrate.
- **Volume mode**: prefer **Block** volume mode over Filesystem for virtual machine disks — it performs closer to bare metal.
- **Snapshots and clone**: creating snapshots (and cloning, which is snapshot-based) requires the disk's StorageClass to have a bound `VolumeSnapshotClass`.

Procedure

1 Enabling Node Virtualization

When the nodes of a self-built cluster are **physical machines**, you can control whether to allow Kubernetes to schedule Virtual Machine Instances (VMIs) on that node by enabling or disabling the node virtualization switch.

- When the switch is enabled, new virtual machines are allowed to be scheduled on the physical machine node; Windows physical nodes do not support enabling virtualization.
- When the switch is disabled, new virtual machines are prevented from being scheduled on the physical machine node, but it does not affect virtual machines that are already running on that node.

Procedure

1. Enter **Administrator**.
2. In the left navigation bar, click **Cluster Management** > **Clusters**.
3. Click ***Self-Built Cluster Name***.
4. On the **Nodes** tab, click the ⋮ to the right of the node where you want to set the virtualization switch > **Enable Virtualization**.
5. Click **Confirm**.

2 Deploying Operator

1. Login, go to the **Administrator** page.
2. Click **Marketplace** > **OperatorHub** to enter the **OperatorHub** page.
3. Find the **ACP Virtualization with KubeVirt**, click **Install**, and navigate to the **Install ACP Virtualization with KubeVirt** page.
   Configuration Parameters:

Parameter	Recommended Configuration
Channel	The default channel is `alpha`.
Installation Mode	`Cluster`: All namespaces in the cluster share a single Operator instance for creation and management, resulting in lower resource usage.
Installation Place	Select `Recommended`, Namespace only support **kubevirt**.
Upgrade Strategy	`Manual`: When there is a new version in the Operator Hub, manual confirmation is required to upgrade the Operator to the latest version.

3 Creating a HyperConverged Instance

1. Enter **Administrator**.
2. Click **Marketplace** > **OperatorHub**.
3. Find the **ACP Virtualization with KubeVirt**, click it to enter the **ACP Virtualization with KubeVirt** detail info page.
4. Click **All Instances**
5. Click **Create Instance** on the **HyperConverged** instance card.
   Note: Only one **HyperConverged** instance needs to be created in each cluster.
6. Switch to YAML view and only replace the ***placeholder*** specified in the `spec.storageImport.insecureRegistries` field in the example with the correct

virtual machine image repository address, for example: `192.168.16.214:60080`, keeping other parameters at their default values.

```
spec:
  storageImport:
    insecureRegistries:
      - placeholder
```

Replacement result:

```
spec:
  storageImport:
    insecureRegistries:
      - '192.168.16.214:60080'
```

7. Click **Create** and wait for the CDI and KubeVirt type instances to be automatically created in the resource list, while ensuring that the **status.phase** displayed in the YAML is `deployed`, indicating that the HyperConverged instance has been successfully created.
   Equivalently, from the command line, apply the `HyperConverged` resource and wait for it to be `Available`:

```
kubectl apply -f - <<'EOF'
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: kubevirt
spec: {}   # defaults are applied by the operator; tune liveMigrat
ionConfig, storageImport, etc. as needed
EOF

kubectl get hyperconverged kubevirt-hyperconverged -n kubevirt \
  -o jsonpath='{range .status.conditions[?(@.type=="Available")]}
{.status}{"\n"}{end}'
```

4 Configuring Virtual Machine Overcommit Ratio (Optional)

- Configuring the overcommit ratio for the cluster where the virtual machines reside in **Cluster Management** > **Clusters**.
- Or Configuring the overcommit ratio for the namespace where the virtual machines are located in **Project Management** > **Namespaces**.

Important Notes

- Virtual machines only support CPU overcommit ratios, and the recommended configuration value is between 2 and 4.
- Once the overcommit ratio is enabled for virtual machines, when creating a virtual machine, the container's request value (requests) is fixed as **specified limit value (limits) / VM overcommit ratio**, making the user's request set through YAML ineffective.
  For example: Assuming the CPU resource overcommit ratio is set to 4 for the virtual machine, if the user specifies a CPU limit value of 4c when creating the virtual machine, the CPU request value would be 4c/4 = 1c.

Resource Quota Explanation

The memory resource quota for virtual machines is limited by the memory resource quota of the namespace they reside in. Since the memory of the Pod hosting the virtual machine is usually larger than the actual available memory of the virtual machine, it is recommended to reserve 20% of the resources. When the remaining available resources in the namespace are below 20%, please promptly scale up the resources.

Images

Introduction

Introduction

Advantages

Uploading Images from Outside the Cluster

Guides

Adding Virtual Machine Images

Procedure

Update/Delete Virtual Machine I

Update/Dele

How To

Uploading a Local Image and C

Prerequisites

Creating Windows Images Base

Prerequisites

Creating Li

Prerequisites

Procedure

Creating a Windows virtual machine

Constraints and Limitations

Procedure

Remote Access

Constraints and

Procedure

Exporting V

Procedure

Permissions

Permissions

Introduction

Alauda Container Platform Virtualization with KubeVirt leverages Kubernetes extended API capabilities to abstract virtual machine images as a **Custom Resource Definition** (CRD). It provides a user interface (UI) for users to easily import virtual machine images stored in remote repositories into ACP for usage.

TOC

Advantages

- Support for Mainstream Operating Systems
  Supports various commonly used Linux distributions and Windows operating systems.
- Multi-Architecture Support
  Compatible with both **X86_64** and **ARM64** architectures.
- Multi-Source Support
  Allows importing virtual machine images from:
  - Image registries
  - File servers

  - S3-compatible object storage

- Multi-Format Support

  Supports virtual machine images in **QCOW2** and **RAW** formats.

Uploading Images from Outside the Cluster

To upload a **local** image file into the platform (rather than importing from a remote source), use `virtctl image-upload`, which streams the file to the CDI upload proxy. The `cdi-uploadproxy` Service (in the `kubevirt` namespace) must be reachable from where you run the command — expose it through the Gateway API, an Ingress, or a `NodePort` / `LoadBalancer` Service so administrators can upload images from outside the cluster. For the full procedure, including exposing the proxy and creating a virtual machine from the uploaded image, see Uploading a Local Image and Creating a Virtual Machine; for the `virtctl` reference, see Using the virtctl Command-Line Tool.

Guides

Adding Virtual Machine Images **Update/Delete Virtual Machine I** **Update/Dele**

Procedure

Adding Virtual Machine Images

The platform supports adding **X86_64** and **ARM64 (Alpha)** architecture virtual machine images, enabling developers to quickly create virtual machines for existing services and facilitate the migration of business systems.

TOC

Procedure

1. Access **Administrator**.
2. In the left navigation bar, click **Virtualization Management** > **Virtual Machine Images**.
3. Click **Add Virtual Machine Image**.
4. Refer to the instructions below to configure the relevant parameters.

Parameter	Description
Provisioning Method	Currently, only **Public Image** method is supported, meaning the added image can be used in assigned projects.

Parameter	Description
Operating System	Supported operating systems include: **CentOS/Ubuntu/RedHat/Debian/TLinux/Other Linux/Windows (Alpha)**. Supported system architectures are: **X86_64** and **ARM64 (Alpha)**.
Source	• **Image Repository**: Virtual machine images stored in a container image repository. • **HTTP**: Virtual machine images stored on a file server using the HTTP protocol. • **Object Storage (S3)**: Virtual machine images that can be retrieved using Object Storage Protocol (S3). If they do not require authentication, please use HTTP as the source.
CPU Architecture	Tag CPU architecture information. For image repository sources, multiple selections are supported; for other sources, only single selection is allowed.
Image Address	Supports KVM virtual machine images, including qcow2/raw formats. • If from an image repository, enter `repository_address:image_version`, e.g., `registry.example.com/library/ubuntu:latest`. • If from an HTTP source, enter the image file URL, which must start with `http://` or `https://`, e.g., `http://192.168.0.1/vm_image/centos_7.8.qcow2`. • If from an Object Storage (S3), enter the image address that can be retrieved via Object Storage Protocol (S3), e.g., `https://endpoint/bucket/centos.qcow2`.
Authentication	Depending on whether the image repository requires authentication, you can toggle the switch on or off. If enabled, you can choose from existing image credentials or click **Add Credentials**, supporting only **Username/Password** type credentials.

Parameter	Description
	Note: When the source is Object Storage (S3), authentication cannot be turned off.
Assigned Project	Assign usage permissions for this image to projects. • All Projects: Assigns usage permissions of the image to all projects. • Specific Project: Assigns usage permissions of the image to a specified project. • No Assignment: Do not assign to any projects for now. After image creation, you can assign it through **Update Image** operation.

5. Click **Add**.

Update/Delete Virtual Machine Images

1. Navigate to **Administrator**.
2. In the left sidebar, click **Virtualization Management** > **Virtual Machine Images**.
3. Click ⋮ > **Update/Delete**.
4. After confirmation, click **Update/Delete**.

Update/Delete Image Credentials

1. Go to **Administrator**.
2. In the left navigation bar, click **Virtualization Management** > **Virtual Machine Images**.
3. In the **Image Credentials** tab, click ⋮ > **Update/Delete**.
4. After confirmation, click **Update/Delete**.

How To

Uploading a Local Image and C

Prerequisites

Procedure

Creating a Windows virtual machine

Creating Windows Images Base

Prerequisites

Constraints and Limitations

Procedure

Remote Access

Creating Li

Prerequisites

Constraints and

Procedure

Exporting V

Procedure

Uploading a Local Image and Creating a Virtual Machine

Use this procedure when you already have a virtual machine disk image **on your local machine** (for example a `qcow2`, `raw`, or `img` file downloaded from an OS vendor) and want to run a virtual machine from it, rather than importing the image from a remote HTTP, registry, or S3 source.

On Alauda Container Platform this is a **command-line workflow** built on the KubeVirt `virtctl` client and the CDI upload proxy — the web console does not yet offer a local-upload source. The uploaded image is stored as a `DataSource` (a **bootable volume**) that virtual machines can boot from and that can be reused by many virtual machines.

> **Tip**: If instead you want to **install** an operating system from an ISO, see Creating Linux Images Based on ISO and Creating Windows Images Based on ISO.

TOC

Prerequisites

- The `virtctl` command-line client is installed and matches the cluster's KubeVirt version. `virtctl` is the standard KubeVirt CLI; download the binary that matches your cluster from the KubeVirt releases and place it on your `PATH`.
- The `kubectl` command-line tool is installed and configured to access the cluster.
- A local disk image in `qcow2`, `raw`, or `img` format. To speed up the upload, compress the image first with `virt-sparsify`, `xz`, or `gzip`.
- A `StorageClass` that supports the `ReadWriteOnce` (RWO) access mode.
- The `cdi-uploadproxy` Service (in the `kubevirt` namespace) is reachable from where you run `virtctl` — see Step 1.

Procedure

1 Expose the CDI upload proxy outside the cluster

`virtctl image-upload` streams the image to the `cdi-uploadproxy` Service, which is an internal `ClusterIP` Service in the `kubevirt` namespace. You must make it reachable from where you run `virtctl`.

> **Note**: On OpenShift, CDI automatically discovers an OpenShift `Route` to `cdi-uploadproxy` and fills in the upload URL for you. Alauda Container Platform runs on upstream Kubernetes, which has no `Route` resource, so you expose the Service yourself (this step) and set the upload URL explicitly (Step 2).

Use **TLS passthrough** so that `virtctl` negotiates TLS directly with `cdi-uploadproxy` (the proxy serves a self-signed certificate, so clients pass `--insecure`). For every method below the backend is the `cdi-uploadproxy` Service on port `443` in the `kubevirt` namespace; note the resulting external URL for Step 2.

- **Gateway API (Envoy Gateway) — recommended.** Add a `TLS` listener in `Passthrough` mode to a `Gateway`, then create a `TLSRoute` whose backend is the

`cdi-uploadproxy` Service on port `443`. The Gateway's external address — a `LoadBalancer` IP or a `NodePort`, set by the gateway's Service Type — becomes the upload URL. See Configure GatewayAPI Gateway and Configure GatewayAPI Route for the full procedure, and Envoy Gateway Operator to install the controller.

- **Ingress (ingress-nginx).** Expose `cdi-uploadproxy` with an SSL-passthrough `Ingress`. See Configure Ingresses and Ingress Nginx Operator. The legacy ALB ingress (`cpaas.io/alb2`) is deprecated.
- **LoadBalancer or NodePort Service.** Without an ingress controller, expose the Service directly — a `LoadBalancer` where MetalLB is configured, or a `NodePort`, which always works:

```
kubectl expose service cdi-uploadproxy -n kubevirt \
  --name=cdi-uploadproxy-np --type=NodePort --port=443 --target-po
rt=8443
kubectl get service cdi-uploadproxy-np -n kubevirt   # note the 44
3:<nodePort> mapping
```

Reach the proxy at `https://<node-ip>:<nodePort>`.

After exposing the Service, verify it answers through the new endpoint (the certificate is self-signed, so use `-k`):

```
curl -k https://<upload-proxy-host>/healthz
# -> OK
```

2 Point `virtctl` at the upload proxy URL

`virtctl image-upload` needs to know the external URL from Step 1. Choose one of the following.

Configure it once (persistent). Set `CDIConfig.spec.uploadProxyURLOverride` so that every `virtctl image-upload` discovers the URL automatically. Because the CDI configuration is managed by the HyperConverged (HCO) operator — which reconciles direct edits away and does not expose this field — set it through HCO's jsonpatch annotation:

```
kubectl annotate hyperconverged kubevirt-hyperconverged -n kubevirt \
  'containerizeddataimporter.kubevirt.io/jsonpatch=[{"op":"add","pat
h":"/spec/config/uploadProxyURLOverride","value":"https://cdi-uploadp
roxy.example.com:8443"}]' \
  --overwrite

# Verify it propagated to the calculated URL that virtctl reads:
kubectl get cdiconfig config -o jsonpath='{.status.uploadProxyURL}
{"\n"}'
```

> **Warning**: HCO documents jsonpatch annotations as an advanced, **unsupported** mechanism — an incorrect patch can destabilize the virtualization stack. Use it deliberately, and remove the annotation (`kubectl annotate hyperconverged kubevirt-hyperconverged -n kubevirt containerizeddataimporter.kubevirt.io/jsonpatch-`) to revert.

Or pass it per command. Skip the cluster configuration and give the URL on each upload with `--uploadproxy-url` (shown in Step 3).

3 Upload the local image

Run `virtctl image-upload`. The `--datasource` flag also creates a `DataSource` that points at the uploaded volume, making it selectable as a bootable volume:

```
virtctl image-upload dv uploaded-image \
  --datasource \
  --size=30Gi \
  --storage-class=rbd-1 \
  --image-path=./my-image.qcow2 \
  --namespace=my-namespace
# add --uploadproxy-url=https://cdi-uploadproxy.example.com:8443 --in
secure
# if you did NOT set uploadProxyURLOverride in Step 2.
```

Notes:

- `--size` must be **larger than the image's virtual disk size** (not the file size). For example, if your image has a 24 GiB virtual disk, request at least a 30 GiB volume.

CDI rejects a volume smaller than the uncompressed disk.

- Use `--no-create` (and omit `--size`) to upload into a `DataVolume` that already exists.
- `--insecure` skips verification of the proxy's self-signed certificate.

Wait for the upload and import to finish:

```
kubectl get dv uploaded-image -n my-namespace -w
# PHASE: UploadReady -> ... -> Succeeded

kubectl get datasource uploaded-image -n my-namespace
# the DataSource becomes Ready
```

> **Note**: A `DataSource` created by `virtctl --datasource` does not carry the `virtualization.cpaas.io/*` labels used to mark a bootable volume. The image is fully usable through the API (by `sourceRef`, as in the next step); to also apply that label contract, see Bootable Volumes.

4 Create a virtual machine from the uploaded image

Reference the `DataSource` from a `dataVolumeTemplates` entry via `sourceRef`. KubeVirt clones the uploaded volume into a fresh disk for the new virtual machine:

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: my-namespace
spec:
  runStrategy: RerunOnFailure
  dataVolumeTemplates:
    - metadata:
        name: example-vm-rootdisk
      spec:
        sourceRef:
          kind: DataSource
          name: uploaded-image
          namespace: my-namespace
        storage:
          resources:
            requests:
              storage: 30Gi
          storageClassName: rbd-1
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
        resources:
          requests:
            memory: 2Gi
        devices:
          disks:
            - name: rootdisk
              disk:
                bus: virtio
          interfaces:
            - name: default
              masquerade: {}
      networks:
        - name: default
          pod: {}
      volumes:
```

```
    - name: rootdisk
      dataVolume:
        name: example-vm-rootdisk
```

```
kubectl apply -f example-vm.yaml
kubectl get vm example-vm -n my-namespace          # STATUS: Running
kubectl get vmi example-vm -n my-namespace         # PHASE: Running,
with an IP
```

Confirm the operating system boots by opening the serial console; a login prompt indicates a successful boot:

```
virtctl console example-vm -n my-namespace
# example-vm login:
```

> **Tip**: The uploaded image can also be referenced as a bootable volume when creating a virtual machine, once it carries the required labels — see Creating Virtual Machines and Bootable Volumes.

Creating a Windows virtual machine

For Windows, upload the image to a volume and then clone it when you create the virtual machine, applying an `autounattend.xml` answer file during first boot. The upload step is identical to the procedure above (use a `qcow2` / `raw` / `img` Windows disk image); for the ISO-based install flow and the VirtIO driver requirements, see Creating Windows Images Based on ISO.

Creating Windows Images Based on ISO using KubeVirt

This document discusses a virtual machine solution based on the open-source component KubeVirt, using KubeVirt virtualization technology to create a Windows operating system image through an ISO image file. The ISO files are uploaded to the cluster via CDI DataVolume, eliminating the need to build and push container images to a registry.

TOC

Prerequisites

- All components in the cluster are functioning correctly.
- Please prepare the Windows image and the latest virtio-win-tools ↗ in advance.
- The `kubectl` command-line tool is installed and configured to access the cluster.
- A StorageClass that supports the `ReadWriteMany` (RWX) access mode is available in the cluster. Live migration requires every PVC attached to the virtual machine to be shared, so all DataVolumes in this document use RWX. If you do not need live migration, `ReadWriteOnce` (RWO) also works.

Constraints and Limitations

- When starting KubeVirt, the size of the custom image's filesystem will affect the speed of writing the image to the disk in PVC. If the filesystem is too large, it may result in extended creation times.
- It is recommended to keep the Windows C drive below 100G to minimize the initial size. Subsequent expansion must be done manually after creation for Windows systems.

Procedure

1 Upload ISO Files to DataVolumes

Upload the Windows ISO and virtio-win ISO files directly to the cluster storage using the CDI upload mechanism.

Set up the CDI Upload Proxy

1. Set up port-forwarding to the CDI upload proxy. This port-forward session will be used for uploading both ISO files. The `cdi-uploadproxy` Service runs in the namespace the virtualization components are installed in, which is `kubevirt` by default.

```
kubectl port-forward -n kubevirt svc/cdi-uploadproxy 8443:443 &
```

Note: Each upload needs its own authentication token, obtained from CDI with an `UploadTokenRequest`. A token authorizes uploading to **one specific PVC** and is valid for **5 minutes**, so the token is requested in each upload procedure below, immediately before the transfer starts, rather than once up front.

Upload the Windows ISO

1. Create an upload-type DataVolume for the Windows ISO by saving the following YAML to a file named `dv-win-iso.yaml`. Adjust the `storage` size based on the actual ISO file size (Windows ISOs are typically 4–6 GB).

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: win-iso-dv
  namespace: default
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
spec:
  source:
    upload: {}
  storage:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 8Gi
    storageClassName: vm-cephrbd  # Replace with your actual Stora
geClass
    volumeMode: Block
```

2. Execute the following command to create the DataVolume.

```bash
kubectl apply -f dv-win-iso.yaml
```

3. Wait for the DataVolume to enter the `UploadReady` phase.

```bash
kubectl get dv win-iso-dv -w
# The PHASE column should show UploadReady
```

4. Request an upload token for the `win-iso-dv` PVC. The token is valid for 5 minutes, so run this immediately before the upload.

```
TOKEN=$(kubectl create -o jsonpath='{.status.token}' -f - <<'EOF'
apiVersion: upload.cdi.kubevirt.io/v1beta1
kind: UploadTokenRequest
metadata:
  name: win-iso-dv-upload
  namespace: default
spec:
  pvcName: win-iso-dv
EOF
)
```

5. Upload the Windows ISO file using curl. Replace the file path with the actual path to your ISO.

```
curl -v --insecure \
  -H "Authorization: Bearer ${TOKEN}" \
  --data-binary @/path/to/en_windows_server_2019_x64_dvd_4cb967d8.iso \
  "https://localhost:8443/v1beta1/upload"
```

6. Verify that the DataVolume status changes to `Succeeded`.

```
kubectl get dv win-iso-dv
# The PHASE column should show Succeeded
```

Upload the virtio-win ISO

1. Create an upload-type DataVolume for the virtio-win ISO by saving the following YAML to a file named `dv-virtio-iso.yaml`.

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: virtio-iso-dv
  namespace: default
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
spec:
  source:
    upload: {}
  storage:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 1Gi
    storageClassName: vm-cephrbd  # Replace with your actual StorageClass
    volumeMode: Block
```

2. Execute the following command to create the DataVolume.

```bash
kubectl apply -f dv-virtio-iso.yaml
```

3. Wait for the DataVolume to enter the `UploadReady` phase.

```bash
kubectl get dv virtio-iso-dv -w
# The PHASE column should show UploadReady
```

4. Request an upload token for the `virtio-iso-dv` PVC. This is a separate token from the one used for the Windows ISO, because a token authorizes uploading to a single PVC.

```
TOKEN=$(kubectl create -o jsonpath='{.status.token}' -f - <<'EOF'
apiVersion: upload.cdi.kubevirt.io/v1beta1
kind: UploadTokenRequest
metadata:
  name: virtio-iso-dv-upload
  namespace: default
spec:
  pvcName: virtio-iso-dv
EOF
)
```

5. Upload the virtio-win ISO file using curl.

```
curl -v --insecure \
  -H "Authorization: Bearer ${TOKEN}" \
  --data-binary @/path/to/virtio-win.iso \
  "https://localhost:8443/v1beta1/upload"
```

6. Verify that the DataVolume status changes to `Succeeded`.

```
kubectl get dv virtio-iso-dv
# The PHASE column should show Succeeded
```

Note: The `--insecure` flag is used to skip self-signed certificate verification. In a production environment, configure proper certificates. Large file uploads may take a long time; ensure network stability.

2 Create Virtual Machine

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Machines**.
3. Click on **Create Virtual Machine**.
4. Fill in the necessary parameters such as **Name**, **Image**, etc., in the form page. For detailed parameters and configuration, please refer to Create Virtual Machine.
5. Switch to YAML.

6. Replace the configuration under the spec.template.spec.domain.devices field with the following content. The USB tablet input device is required for correct mouse positioning in the VNC console (see kubevirt#2392 ↗, kubevirt#3474 ↗). Without it, the mouse pointer will be misaligned because VNC uses absolute coordinates while the default PS/2 mouse only supports relative coordinates.

```
domain:
  devices:
    inputs:
      - type: tablet
        bus: usb
        name: tablet
    disks:
      - disk:
          bus: virtio
        name: cloudinitdisk
      - bootOrder: 1
        cdrom:
          bus: sata
        name: win-iso
      - cdrom:
          bus: sata
        name: virtio-iso
      - disk:
          bus: sata
        name: rootfs
        bootOrder: 10
```

7. Replace the spec.template.spec.volumes field with the following content. Both ISO files are referenced as DataVolumes instead of container images.

```
      volumes:
        - cloudInitConfigDrive:
            userData: >-
              #cloud-config
              disable_root: false
              ssh_pwauth: true
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # Generate with: mkpass
wd --method=SHA-512 --rounds=4096
          name: cloudinitdisk
        - dataVolume:
            name: win-iso-dv
          name: win-iso
        - dataVolume:
            name: virtio-iso-dv
          name: virtio-iso
        - dataVolume:
            name: aa-test-rootfs
          name: rootfs
```

8. Add the following annotation under `spec.template.metadata.annotations`. The primary interface of this virtual machine uses `bridge` binding on the pod network, which KubeVirt refuses to live-migrate by default; this annotation declares that the underlying CNI can follow the migration and lifts that restriction. Kube-OVN, the default CNI of the platform, preserves the virtual machine IP address across a migration. Without this annotation, migration is rejected with `InterfaceNotLiveMigratable`.

   ```
   template:
     metadata:
       annotations:
         kubevirt.io/allow-pod-bridge-network-live-migration: "tru
e"
   ```

   Note: KubeVirt only reports the first condition that blocks live migration, and the interface check runs before the disk check. If any PVC still uses `ReadWriteOnce`, the virtual machine remains non-migratable after adding this annotation, and the

reason reported changes to `DisksNotLiveMigratable`. Both the annotation and RWX access modes are required.

9. Check the YAML file. The complete YAML after finishing the configuration is as follows.

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    cpaas.io/creator: test@example.io
    cpaas.io/display-name: ""
    cpaas.io/updated-at: 2024-09-01T14:57:55Z
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  generation: 16
  labels:
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa-test
  name: aa-test
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        labels:
          vm.cpaas.io/reclaim-policy: Delete
          vm.cpaas.io/used-by: aa-test
        name: aa-test-rootfs
      spec:
        pvc:
          accessModes:
            - ReadWriteMany
          resources:
            requests:
              storage: 100Gi
          storageClassName: vm-cephrbd
          volumeMode: Block
        source:
          blank: {}
  running: true
  template:
    metadata:
      annotations:
        cpaas.io/creator: test@example.io
        cpaas.io/display-name: ""
```

```
        cpaas.io/updated-at: 2024-09-01T14:55:44Z
        kubevirt.io/latest-observed-api-version: v1
        kubevirt.io/storage-observed-api-version: v1
        kubevirt.io/allow-pod-bridge-network-live-migration: "tru
e"
      creationTimestamp: null
      labels:
        virtualization.cpaas.io/image-name: debian-2120-x86
        virtualization.cpaas.io/image-os-arch: amd64
        virtualization.cpaas.io/image-os-type: debian
        virtualization.cpaas.io/image-supply-by: public
        vm.cpaas.io/name: aa-test
    spec:
      affinity:
        nodeAffinity: {}
      architecture: amd64
      domain:
        devices:
          inputs:
            - type: tablet
              bus: usb
              name: tablet
          disks:
            - disk:
                bus: virtio
              name: cloudinitdisk
            - bootOrder: 1
              cdrom:
                bus: sata
              name: win-iso
            - cdrom:
                bus: sata
              name: virtio-iso
            - disk:
                bus: sata
              name: rootfs
              bootOrder: 10
          interfaces:
            - bridge: {}
              name: default
        machine:
          type: q35
        resources:
          limits:
```

```
            cpu: "4"
            memory: 8Gi
          requests:
            cpu: "4"
            memory: 8Gi
      networks:
        - name: default
          pod: {}
      nodeSelector:
        kubernetes.io/arch: amd64
        vm.cpaas.io/baremetal: "true"
      volumes:
        - cloudInitConfigDrive:
            userData: >-
              #cloud-config
              disable_root: false
              ssh_pwauth: true
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # Generate with: mkpass
wd --method=SHA-512 --rounds=4096
          name: cloudinitdisk
        - dataVolume:
            name: win-iso-dv
          name: win-iso
        - dataVolume:
            name: virtio-iso-dv
          name: virtio-iso
        - dataVolume:
            name: aa-test-rootfs
          name: rootfs
```

10. Click **Create**.
11. Click **Actions** > **VNC Login**.
12. When the prompt **press any key boot from CD or DVD** appears, press any key to enter the Windows installation program; if you do not see the prompt, click on **Send Remote Command** in the top left of the page, then select **Ctrl-Alt-Delete** from the dropdown menu to restart the server.

    Note: If a message appears at the top of the virtual machine details page stating **The current virtual machine has configuration changes that require a restart to take**

effect, please restart, this message can be ignored; no restart is necessary.

3 Install Windows Operating System

1. Follow the installation instructions to install the system after entering the installation page.
   Note: During the partition selection step, the bus must be sata for the disk to be correctly recognized. Therefore, you need to select each partition in turn and click **Delete** to remove all partitions, allowing the system to handle it automatically.
2. After configuring the administrator account password, click **Send Remote Command** in the top left of the page, then select **Ctrl-Alt-Delete** from the dropdown menu.
3. When prompted **The Ctrl+Alt+Delete combination will restart the server, confirm to restart**, click **OK**.
4. Enter the password to access the Windows system desktop; at this point, the Windows operating system installation is complete.

4 Install virtio-win-tools

This tool primarily contains the necessary drivers.

1. Open File Explorer.
2. Double-click **CD Drive(E:) virtio-win-<version>**, run the **virtio-win-guest-tools** directory to enter the installation page, and follow the installation instructions. The <version> part should be based on the actual situation.
3. After the installation is complete, power off the Windows system.

5 Export Custom Windows Image

Please refer to Export Virtual Machine Image for the specific operation.

6 Use Windows Image

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Machines**.
3. Click on **Create Virtual Machine**.

4. Fill in the necessary parameters on the form page. For the image, select the exported Windows image. For detailed parameters and configuration, please refer to Create Virtual Machine.
5. (Optional) If using a newer operating system, such as Windows 11, enable features like clock, UEFI, TPM, etc. Switch to YAML and replace the original YAML file with the following YAML file.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-windows
  name: vmi-windows
spec:
  domain:
    clock:
      timer:
        hpet:
          present: false
        hyperv: {}
        pit:
          tickPolicy: delay
        rtc:
          tickPolicy: catchup
      utc: {}
    cpu:
      cores: 2
    devices:
      inputs:
        - type: tablet
          bus: usb
          name: tablet
      disks:
      - disk:
          bus: sata
        name: pvcdisk
      interfaces:
      - masquerade: {}
        model: e1000
        name: default
      tpm: {}
    features:
      acpi: {}
      apic: {}
      hyperv:
        relaxed: {}
        spinlocks:
          spinlocks: 8191
        vapic: {}
      smm: {}
```

```
    firmware:
      bootloader:
        efi:
          secureBoot: true
      uuid: 5d307ca9-b3ef-428c-8861-06e72d69f223
    resources:
      requests:
        memory: 4Gi
  networks:
  - name: default
    pod: {}
  terminationGracePeriodSeconds: 0
  volumes:
  - name: pvcdisk
    persistentVolumeClaim:
      claimName: disk-windows
  - name: winiso
    persistentVolumeClaim:
      claimName: win11cd-pvc
```

6. Click **Create**.

7 Add Internal Route

By configuring a NodePort type internal route, expose the port for remote desktop connections.

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Machines**.
3. Click on the virtual machine name created with the Windows image in the list to enter the details page.
4. Click on the **Add** icon next to **Internal Route** in the **Login Information** area.
5. Configure parameters according to the following instructions.

Parameter	Description
Type	Select **NodePort**.
Port	• Protocol: Select TCP.

Parameter	Description
	• Service Port: Use 3389. • Virtual Machine Port: Use 3389. • Service Port Name: Use rdp.

6. Click **OK** to return to the details page.
7. Click on the **Internal Route** link in the **Login Information** area.
8. Save the **Virtual IP** information in the basic information area and the **Host Port** information in the port area.

Remote Access

This document discusses using the Windows operating system for remote connection as an example. Other operating systems can use software that supports the RDP protocol for connection.

1. Open **Remote Desktop Connection**.
2. Enter the saved Virtual IP and Host Port from the **Add Internal Route** step, formatted as **Virtual IP:Host Port**, for example: 192.1.1.1:3389 .
3. Click **Connect**.

Creating Linux Images Based on ISO Using KubeVirt

This document describes a virtual machine solution implemented based on the open-source component KubeVirt. It utilizes KubeVirt virtualization technology to create a Linux operating system image from an ISO image file. The ISO is uploaded to the cluster via CDI DataVolume, eliminating the need to build and push container images to a registry.

TOC

Prerequisites

- All components in the cluster are functioning properly.
- A Linux image should be prepared in advance. This document uses the Ubuntu operating system ↗ as an example.
- The `kubectl` command-line tool is installed and configured to access the cluster.
- A StorageClass that supports `ReadWriteOnce` (RWO) access mode is available in the cluster.

Constraints and Limitations

- When starting KubeVirt, the file system size of the custom image will affect the speed of writing the image to the PVC disk. If the file system is too large, it may result in a prolonged creation time.
- It is recommended to keep the Linux root partition size below 100G to minimize the initial size. After configuring cloud-init, allocate larger storage for the root partition when creating the virtual machine, and the system will automatically expand it.

Procedure

1 Upload the Linux ISO to a DataVolume

Upload the ISO file directly to the cluster storage using the CDI upload mechanism, without building a container image.

1. Create an upload-type DataVolume by saving the following YAML to a file named `dv-iso-upload.yaml`. Adjust the `storage` size based on the actual ISO file size.

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: iso-upload-dv
  namespace: default
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
spec:
  source:
    upload: {}
  storage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 8Gi
    storageClassName: vm-cephrbd  # Replace with your actual StorageClass
    volumeMode: Block
```

2. Execute the following command to create the DataVolume.

```bash
kubectl apply -f dv-iso-upload.yaml
```

3. Wait for the DataVolume to enter the `UploadReady` phase.

```bash
kubectl get dv iso-upload-dv -w
# The PHASE column should show UploadReady
```

4. Set up port-forwarding to the CDI upload proxy. The `cdi-uploadproxy` Service runs in the namespace the virtualization components are installed in, which is `kubevirt` by default.

```bash
kubectl port-forward -n kubevirt svc/cdi-uploadproxy 8443:443 &
```

5. Request an upload token from CDI with an `UploadTokenRequest`. The token authorizes uploading to the `iso-upload-dv` PVC and is valid for 5 minutes, so run this immediately before the upload.

```
TOKEN=$(kubectl create -o jsonpath='{.status.token}' -f - <<'EOF'
apiVersion: upload.cdi.kubevirt.io/v1beta1
kind: UploadTokenRequest
metadata:
  name: iso-upload-dv-upload
  namespace: default
spec:
  pvcName: iso-upload-dv
EOF
)
```

6. Upload the ISO file using curl. Replace the file path with the actual path to your ISO.

```
curl -v --insecure \
  -H "Authorization: Bearer ${TOKEN}" \
  --data-binary @/path/to/ubuntu-24.04-live-server-amd64.iso \
  "https://localhost:8443/v1beta1/upload"
```

Note: The `--insecure` flag is used to skip self-signed certificate verification. In a production environment, configure proper certificates. Large file uploads may take a long time; ensure network stability.

7. Verify that the DataVolume status changes to `Succeeded`.

```
kubectl get dv iso-upload-dv
# The PHASE column should show Succeeded
```

2 Create Virtual Machine

1. Enter the **Container Platform**.
2. Click **Virtualization** > **Virtual Machines** in the left navigation bar.
3. Click **Create Virtual Machine**.
4. Fill in the parameters on the form page as follows. For specific parameters and configurations, please refer to Create Virtual Machine.

Parameter	Description
Select Image	Choose the template image for the virtual machine.
IP Address	Keep default, which will be obtained via **DHCP**.
Network Mode	Use **NAT** mode; do not use **bridged** mode here.

5. Switch to YAML.
6. Replace the configuration under the spec.template.spec.domain.devices.disks field with the following content. The ISO DataVolume is mounted as a CDROM with the highest boot priority, and the rootfs disk is used as the installation target.

```
domain:
  devices:
    disks:
      - bootOrder: 1
        cdrom:
          bus: sata
        name: iso-disk
      - disk:
          bus: virtio
        name: cloudinitdisk
      - disk:
          bus: virtio
        name: rootfs
        bootOrder: 10
```

7. Replace the spec.template.spec.volumes field with the following content. The ISO is referenced as a DataVolume instead of a container image.

```
      volumes:
        - dataVolume:
            name: iso-upload-dv
          name: iso-disk
        - cloudInitConfigDrive:
            userData: |-
              #cloud-config
              disable_root: false
              ssh_pwauth: false
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # Generate with: mkpass
wd --method=SHA-512 --rounds=4096
          name: cloudinitdisk
        - dataVolume:
            name: aa-rootfs
          name: rootfs
```

8. Review the YAML file; the complete YAML configuration after completion is as follows.

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  labels:
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa
  name: aa
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        labels:
          vm.cpaas.io/reclaim-policy: Delete
          vm.cpaas.io/used-by: aa
        name: aa-rootfs
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 100Gi
          storageClassName: vm-cephrbd
          volumeMode: Block
        source:
          blank: {}
  running: true
  template:
    metadata:
      annotations:
        cpaas.io/creator: test@example.io
        cpaas.io/display-name: ""
        cpaas.io/updated-at: 2024-09-09T03:49:08Z
        kubevirt.io/latest-observed-api-version: v1
        kubevirt.io/storage-observed-api-version: v1
      creationTimestamp: null
      labels:
```

```
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa
spec:
  accessCredentials:
    - sshPublicKey:
        propagationMethod:
          qemuGuestAgent:
            users:
              - root
        source:
          secret:
            secretName: test-xeon
  affinity:
    nodeAffinity: {}
  architecture: amd64
  domain:
    devices:
      disks:
        - bootOrder: 1
          cdrom:
            bus: sata
          name: iso-disk
        - disk:
            bus: virtio
          name: cloudinitdisk
        - disk:
            bus: virtio
          name: rootfs
          bootOrder: 10
      interfaces:
        - bridge: {}
          name: default
    machine:
      type: q35
    resources:
      limits:
        cpu: "1"
        memory: 2Gi
      requests:
        cpu: "1"
        memory: 2Gi
```

```
      networks:
        - name: default
          pod: {}
      nodeSelector:
        kubernetes.io/arch: amd64
        vm.cpaas.io/baremetal: "true"
      volumes:
        - dataVolume:
            name: iso-upload-dv
          name: iso-disk
        - cloudInitConfigDrive:
            userData: |-
              #cloud-config
              disable_root: false
              ssh_pwauth: false
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # Generate with: mkpass
wd --method=SHA-512 --rounds=4096
          name: cloudinitdisk
        - dataVolume:
            name: aa-rootfs
          name: rootfs
```

9. Click **Create**.
10. Click **Actions** > **VNC Login**.
11. When prompted with **press any key boot from CD or DVD**, press any key to enter the installation program; if you do not see the prompt, click **Send Remote Command** in the upper left corner of the page, and then click **Ctrl-Alt-Delete** from the dropdown menu to reboot the server.

    Note: If a message appears at the top of the virtual machine detail page stating **Current virtual machine has configuration changes that require a restart to take effect. Please restart.**, you can ignore this message; a restart is not necessary.

3 Install Linux Operating System

1. After entering the installation page, follow the installation guide to proceed. This document gives an example of installing the Ubuntu operating system; the configuration items during the installation process of different operating systems are

generally similar, and thus will not be elaborated further. Some configuration items are explained below.

Configuration	Description
Installation Type	It is recommended to use a minimal installation to minimize the image size.
Storage Configuration	Choose custom storage. Format the disk to ext4 or xfs format and mount it to the root partition (/). **Note**: Do not use LVM for disk partitioning (Create volume group (LVM)).
SSH Configuration	Choose to install the OpenSSH tools for SSH access.

2. Wait for the installation to complete.

4 Modify YAML File

1. Enter the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click on the **Virtual Machine Name** in the list to enter the details page.
4. Click **Stop**.
5. Click **Actions** > **Update** in the upper right corner.
6. Switch to YAML.
7. Confirm that the disk named **rootfs** under spec.template.spec.domain.devices.disks has a bootOrder of 1. If it is not 1, modify it to 1.
8. Delete the relevant content for the ISO disk named **iso-disk** under spec.template.spec.domain.devices.disks; the specific content to delete is as follows.

```
        - bootOrder: 1
          cdrom:
            bus: sata
          name: iso-disk
```

9. Delete the relevant content for the volume named **iso-disk** under spec.template.spec.volumes; the specific content to delete is as follows.

```
        - dataVolume:
            name: iso-upload-dv
          name: iso-disk
```

10. Click **Update**.
11. Click **Start**.

5 Install Required Software and Modify Configuration

Note: The following commands and configuration files may vary slightly between different operating systems; please adjust according to your actual environment.

1. Enter your username and password to log in to the operating system.
2. Switch to root user privileges.
3. Install the software packages.
   - For CentOS series, execute the command:

     ```
     yum install cloud-utils cloud-init qemu-guest-agent vim
     ```

   - For Debian series, execute the command:

     ```
     apt install cloud-init cloud-guest-utils qemu-guest-agent vim
     ```

4. Edit the SSHD configuration file.
   1. Execute the following command to edit the sshd_config file.

      ```
      vim /etc/ssh/sshd_config
      ```

   2. Add the following configurations.

```
PermitRootLogin yes  # Allow the root user to log in with a pass
word
PubkeyAuthentication yes  # Allow key-based login
```

3. Save the modified configuration.

5. Execute the following command to delete the default password for the root user.

```
passwd -d root
```

6. Modify the source address file.

   1. Execute the following command to modify the system's source address file and change the address to a suitable mirror site address.

      ```
      vim /etc/apt/sources.list.d/ubuntu.sources
      ```

   2. Save the configuration after modifications.

7. Modify the cloud-init configuration to automatically expand the root directory.

   1. Execute the following command to edit the cloud.cfg configuration file.

      ```
      vim /etc/cloud/cloud.cfg
      ```

   2. Add the following configuration content.

      ```
      runcmd:
        - [growpart, /dev/vda, 1]  # The growpart command is used to e
      xtend the partition on the disk, which will extend the /dev/vda1
      partition.
        - [xfs_growfs, /dev/vda1]  # The xfs_growfs command is used to
      extend the XFS file system to occupy all available space on the
      partition. /dev/vda1 is the partition where the file system to b
      e extended is located. After extending the partition, using xfs_
      growfs ensures that the file system itself is also expanded to t
      he new partition size.
      ```

   3. Save the configuration after modifications.

8. After completing the configuration, shut down the operating system.

6 Export and Use the Custom Linux Image

For specific operations, please refer to Export Virtual Machine Image.

Exporting Virtual Machine Images

This feature is used to export the system image of a virtual machine and upload it to object storage, allowing the files in object storage to be added as sources to the platform's virtual machine images.

TOC

Procedure

Note: All operations below must be performed on the control node of the cluster where the virtual machine resides.

1 Stopping the Virtual Machine

1. Go to **Administrator**.

2. In the left navigation bar, click **Virtualization Management** > **Virtual Machines**.
3. Click on the name of the virtual machine whose system image needs to be exported, which will redirect you to the virtual machine details page in the Container Platform.
4. Click **Stop**.

2 Creating the vmexport Resource

1. Open the CLI tool.
2. Execute the following command to set variables.

```
NAMESPACE=<namespace>
VM_NAME=<vm_name>
TTL_DURATION=2h
```

Parameter explanation:

- **NAMESPACE**: The name of the namespace where the virtual machine resides; replace the *<namespace>* part with this name.
- **VM_NAME**: The name of the virtual machine whose system image needs to be exported; replace the *<vm_name>* part with this name.
- **TTL_DURATION**: The lifetime of the export task, defaulting to 2 hours but can be increased as needed.

3. Execute the following command to create the vmexport resource.

```
cat <<EOF | kubectl create -f -
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: export-$VM_NAME
  namespace: $NAMESPACE
spec:
  ttlDuration: $TTL_DURATION
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: $VM_NAME
EOF
```

If similar echo information appears, it indicates successful creation.

```
virtualmachineexport.export.kubevirt.io/export-k1 created
```

4. Execute the following command to check the status of the vmexport resource.

```
kubectl -n $NAMESPACE get vmexport export-$VM_NAME -w
```

Echo information:

```
NAME         SOURCEKIND       SOURCENAME   PHASE
export-k1    VirtualMachine   k1           Ready
```

5. When the PHASE field in the echo information changes to Ready, type ctrl (control) + c to stop the watch operation.
6. Execute the following command to get the TOKEN.

```
TOKEN=$(kubectl -n $NAMESPACE get secret export-token-export-$VM_N
AME -o jsonpath={.data.token} | base64 -d)
```

3 Downloading the Virtual Machine Image File

1. Execute the following command to get the IP address of the virtual machine export Pod in the specified namespace and store it in the EXPORT_SERVER_IP environment variable.

```
EXPORT_SERVER_IP=$(kubectl -n $NAMESPACE get po virt-export-export
-$VM_NAME -o jsonpath='{.status.podIP}')
```

2. Execute the following command to set the URL environment variable, which points to the virtual machine's disk image file.

```
URL=https://$EXPORT_SERVER_IP:8443/volumes/$VM_NAME-rootfs/disk.im
g.gz
```

3. Execute the following command to download the image file, with the downloaded file named disk.img.gz.

```
curl -k -O -H "x-kubevirt-export-token: $TOKEN" $URL
```

4 Uploading the Virtual Machine Image File to Object Storage

Upload the downloaded image file to object storage. Any S3 tool can be used for the upload, and this document will introduce the mc (minio-client) tool as an example.

1. Execute the following command to configure the mc tool and connect to the specified S3 storage service.

```
mc alias set minio <ENDPOINT> <ACCESSKEY> <SECRETKEY>
```

   Parameter explanation:

   - ENDPOINT: The address of the S3 storage service; replace the `<ENDPOINT>` part with this address.
   - ACCESSKEY, SECRETKEY: The user ak and sk of the S3 storage service used for authentication; for related information, please refer to MinIO Object Storage ↗.

2. Execute the following command to create a bucket for storing the virtual machine image files.

```
mc mb minio/vmdisks
```

3. Execute the following command to upload the exported virtual machine image file disk.img.gz to the created bucket.

```
mc put disk.img.gz minio/vmdisks
```

5 Creating the Virtual Machine Image

1. Go to **Administrator**.

2. In the left navigation bar, click **Virtualization Management** > **Virtual Machine Images**.
3. Click **Add Virtual Machine Image**.
4. In the image address, fill in <*ENDPOINT*>/vmdisks/disk.img.gz, replacing the <ENDPOINT> part with the address of the S3 storage service. For other parameter explanations, please refer to Adding Virtual Machine Images.
5. Click **Add**.

Permissions

Function	Action	Platform Administrator	Platform auditors	Pro Ma
virtualmachineimagetemplates `acp-virtualmachineimagetemplates`	View	✓	✓	
	Create	✓	✗	
	Update	✓	✗	
	Delete	✓	✗	

Virtual Machine

Introduction

Introduction

Bootable Volumes

Bootable Volumes

Notes

View bootable volumes

Add a bootable volume

Create a virtual machine from a bootable volume

Update or delete a bootable volume

Guides

Creating Virtual Machines

Prerequisites

Notes

Create Virtual Machine

Using the API

Batch Operations on Virtual Ma

Procedure

Using the API

Pausing an

Notes

Pause a virtual

Resume a virtu

Cloning a Virtual Machine

Logging into the Virtual Machine using VNC

Procedure

Using the API

Managing Virtual Machines

Reset Password

Update Key

Update Specifications

Live Migration

Update NAT Network Configuration

Update Tags and Annotations

Add Service

Reinstall Operating System

Configure IP

Using the API

Imported Virtual Machines Using an InstanceType Spec

Monitoring and Alerts

Monitoring

Alerts

Using the API

Quick Location of Virtual Machines

Prerequisites

Procedure

Using the API

How To

Using the virtctl Command-Line

- Consoles and access
- Uploading and exporting images
- Hot-plugging disks
- Power and migration
- Editing and sealing images with libguestfs

Configuring USB host passthro

- Feature Overview
- Use Cases
- Prerequisites
- Steps
- Operation Result
- Learn More

Installing G

- QEMU guest ag
- VirtIO drivers (V
- Virtual TPM (vT

Virtual Mach

ettin

CPU and Memory Hotplug

- How It Works
- Prerequisites
- Step 1: Create a VM with Hotplug Headroom
- Step 2: Hotplug CPU
- Step 3: Hotplug Memory
- Scaling Down: What Works Live and What
- Verified Behavior Summary
- Troubleshooting

Virtual Machine Recovery

- Steps to Operate

Clone Virtual Machines on KubeVirt

- Ensure Prerequisites
- Start Quickly
- Understand the VirtualMachineClone Obje

Migrate Virt

- Prerequisites
- Prepare the Mig
- Cold Migration

chi

Create a VM Template from an Existing Virtual Machine

- Prerequisites
- Procedure

Component Overview

- Flow of events during fencing and remediation
- Procedure

Troubleshooting

Introduction

KubeVirt provides CRDs (Custom Resource Definitions) such as **VirtualMachine** and **VirtualMachineInstance** to abstract virtual machine (VM) resources. Based on these CRDs, users gain comprehensive VM management capabilities. Building on this foundation, **ACP Virtualization With KubeVirt** further enhances usability by offering a **Web Console**, enabling users to perform various operations with greater ease.

Bootable Volumes

A **bootable volume** is a ready-to-use boot image that virtual machines can be created from. Each bootable volume is a KubeVirt `DataSource` (`cdi.kubevirt.io/v1beta1`) backed by a PVC that CDI populates from an import or clone source. Bootable volumes are aggregated across namespaces — including the shared `kube-public` namespace where platform "golden images" live — so they can be reused by many virtual machines.

When you create a virtual machine and choose **Provision Method** > **Image Instance**, the picker lists the bootable volumes you can boot from.

TOC

Notes

- A bootable volume becomes selectable for virtual-machine creation only when its `DataSource` carries the `virtualization.cpaas.io/*` label contract (operating system, architecture, capacity, storage class, volume mode, access mode). A data source without

these labels still appears in the management list but is **not** offered in the **Image Instance** picker.

- The volume's **Status** is `Importing` while CDI populates the backing PVC and `Ready` once it can be used.
- Local **Upload** as a source is not yet supported; use HTTP/HTTPS, Registry, Clone PVC, or Clone snapshot.

View bootable volumes

Bootable volumes are backed by `DataSource` objects and are aggregated across namespaces (including the shared `kube-public` namespace). List them directly:

```
kubectl get datasource -A
```

Add a bootable volume

A bootable volume supports four source types:

Source type	What it does	DataVolume source
HTTP/HTTPS (default)	Import a qcow2/raw image from an HTTP(S) URL	`http`
Registry	Pull a container disk from a container registry	`registry`
Clone PVC	Duplicate an existing PVC	`pvc`
Clone snapshot	Restore from a VolumeSnapshot	`snapshot`

> **Note**: Create bootable volumes in your **own project namespace**. The shared `kube-public` namespace holds platform **golden images** published by a cluster administrator — ordinary project users can **consume** (clone and boot from) `kube-public` images out of

> the box, but creating or modifying objects in `kube-public` requires elevated platform permissions.

Adding a bootable volume creates two objects: a `DataVolume` (CDI imports the image into a PVC) and a `DataSource` that points at it and carries the label contract. The example imports an image over HTTP:

```yaml
# 1) DataVolume — CDI imports the image into a PVC of the same name.
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: ubuntu-2204
  namespace: demo
spec:
  source:
    http:
      url: https://images.example.com/ubuntu-22.04.qcow2
      # secretRef: my-pull-secret   # for an authenticated source
  storage:
    resources:
      requests:
        storage: 20Gi
    storageClassName: sc-topolvm
    volumeMode: Filesystem
    accessModes:
      - ReadWriteOnce
---
# 2) DataSource — makes the PVC selectable as a bootable volume.
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataSource
metadata:
  name: ubuntu-2204
  namespace: demo
  labels:
    virtualization.cpaas.io/image-os-type: ubuntu
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/size: 20Gi
    virtualization.cpaas.io/storage-class: sc-topolvm
    virtualization.cpaas.io/volume-mode: Filesystem
    virtualization.cpaas.io/access-mode: ReadWriteOnce
    virtualization.cpaas.io/source-origin: url   # url | registry | pvc |
snapshot
spec:
  source:
    pvc:
      name: ubuntu-2204
      namespace: demo
```

For the other source types, replace `spec.source` on the `DataVolume`:

```
# Registry
source:
  registry:
    url: docker://registry.example.com/library/fedora:40
# Clone an existing PVC
source:
  pvc:
    name: source-pvc
    namespace: demo
# Restore from a VolumeSnapshot
source:
  snapshot:
    name: source-snapshot
    namespace: demo
```

Apply and watch the import:

```
kubectl apply -f bootable-volume.yaml
kubectl get datavolume ubuntu-2204 -n demo -w
```

Create a virtual machine from a bootable volume

A virtual machine created from a bootable volume defaults to the bootable volume's namespace (a same-namespace clone).

Under the hood, the virtual machine references the data source in a `dataVolumeTemplates` entry — CDI clones the bootable volume into the new virtual machine's root disk:

```
spec:
  dataVolumeTemplates:
    - metadata:
        name: web-01-rootfs
      spec:
        sourceRef:
          kind: DataSource
          name: ubuntu-2204
          # The bootable volume's namespace. Use kube-public to boot from
an
          # administrator-published golden image.
          namespace: demo
        storage:
          resources:
            requests:
              storage: 20Gi
          storageClassName: sc-topolvm
          volumeMode: Filesystem
          accessModes:
            - ReadWriteOnce
```

The clone source namespace is restricted to the current/target namespace and `kube-public`. Cloning from another non-public namespace may require additional CDI clone permissions.

Update or delete a bootable volume

Deleting a bootable volume removes its `DataSource` and the managed `DataVolume`. The backing PVC is owned by the `DataVolume` (via `ownerReferences`), so it is garbage-collected automatically when the `DataVolume` is deleted — do not delete the PVC while the `DataVolume` still exists, or CDI will re-provision it. The `DataSource`, `DataVolume`, and PVC share the same name:

```
kubectl delete datasource ubuntu-2204 -n demo
kubectl delete datavolume ubuntu-2204 -n demo
```

If CDI already garbage-collected the `DataVolume` after a successful import, `kubectl delete datavolume` returns `NotFound` and the PVC is left without an owner — delete it directly in that case:

```
kubectl delete pvc ubuntu-2204 -n demo
```

Guides

Creating Virtual Machines

- Prerequisites
- Notes
- Create Virtual Machine
- Using the API

Batch Operations on Virtual Ma

- Procedure
- Using the API

Pausing an

- Notes
- Pause a virtual
- Resume a virtu

Cloning a Virtual Machine

Logging into the Virtual Machine using VNC

- Procedure
- Using the API

Managing Virtual Machines

- Reset Password
- Update Key
- Update Specifications
- Live Migration
- Update NAT Network Configuration
- Update Tags and Annotations
- Add Service
- Reinstall Operating System
- Configure IP

Monitoring and Alerts

- Monitoring
- Alerts
- Using the API

Quick Location of Virtual Machines

- Prerequisites
- Procedure

Using the API

Creating Virtual Machines

Create a virtual machine (a `VirtualMachine` resource) using an image, and schedule it to physical nodes with KubeVirt components installed and virtualization enabled. A `VirtualMachine` is the desired-state object (analogous to a Deployment); when it is started, KubeVirt produces a running `VirtualMachineInstance` (analogous to a Pod). For the terms used on this page, see Concepts and Glossary.

You can create a virtual machine through Create Virtual Machine using a form, and you may also switch to YAML format for the operation.

TOC

Prerequisites

- Before creating a virtual machine using an image, please confirm the following with the platform administrator:

- The target cluster is a self-built cluster, and the Kubevirt components have been deployed.
- The target node must be a physical node with virtualization enabled.
- A virtual machine image has been added to the platform.

- If you need to use the physical GPU passthrough feature of the virtual machine, please contact the platform administrator for the following configuration:

  1. Obtain GPU passthrough environment preparation plan and prepare the necessary environment.
  2. Prepare the required physical GPU and enable the related features for physical GPU passthrough for the virtual machine.

Notes

When using Windows virtual machines, only logins via the **username/password** set in the virtual machine image are supported. Please contact the platform administrator to obtain this information in advance.

Create Virtual Machine

Procedure

Note: The following content presents an example of creating a virtual machine using a form, and you may also switch to YAML format for the operation.

1. Enter **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click **Create Virtual Machine**.
4. In the **Basic Information** area, fill in the name and display name of the virtual machine and set tags or annotations.

Parameter	Description
Tags	Used to select objects and find collections of objects that meet certain criteria. Must be a key-value pair, for example: *app.kubernetes.io/name: hello-app*.
Annotations	Used to provide any information to development and operations teams. Must be a key-value pair, for example: *cpaas.io/maintainer: kim*.

5. Set the machine type and choose a virtual machine image.

Parameter	Description
Specifications	You can select recommended usage scenarios or custom resource limits based on your needs.
Physical GPU (Alpha)	Select the model of the physical GPU; only one physical GPU can be allocated to each virtual machine. **Note**: Physical GPU passthrough for the virtual machine refers to the direct allocation of the actual Graphics Processing Unit (GPU) to the virtual machine in a virtualization environment, enabling it to directly access and utilize the physical GPU to achieve graphical performance equivalent to running directly on a physical machine, avoiding performance bottlenecks caused by virtual graphics adapters and enhancing overall performance.
Image	Choose a public image that has been assigned to the platform project by the platform administrator. **Note**: Only supports selecting images with the same **CPU architecture** as the cluster architecture.

6. In the **Storage** area, refer to the following instructions to configure the relevant information.

Parameter	Description
Disk Name	The name of the storage disk; the system disk name cannot be modified.

Parameter	Description
Type	• **Root Disk**: The system automatically creates a VirtIO type rootfs system disk to store the operating system and data. • **Data Disk**: Click to add multiple data disks for persistent data storage. Defaults to VirtIO device. **Note**: Data disk names must not duplicate existing disk names.
Volume Mode	• **File System**: Mount the disk as a mounted file directory. • **Block Device**: Mount the disk as a block device.
Storage Class	The platform maintains virtual machine disks by creating and managing persistent volume claims. You need to specify a storage class required for dynamically creating persistent volume claims. Different storage classes support different volume modes; if there is no available storage class for the selected volume mode, please contact an administrator for addition.
Capacity	The capacity required for the virtual machine storage; the minimum for the system disk is 20 G.
Delete with VM	Defaults to enabled, cannot be modified, indicating that the disk data will also be deleted when the virtual machine is deleted.

7. In the **Network** area, refer to the following instructions to configure the relevant information.

Parameter	Description
IP Address	• Defaults to **Dynamic (DHCP)**; an IP is dynamically assigned when the virtual machine starts, and the IP is released when the virtual machine stops. • If binding a **Static IP**, the virtual machine will always use this IP address even after a restart. If there are no available IPs in the current

Parameter	Description
	project, please release an IP appropriately first.
Network Mode	• **Bridged**: The virtual machine shares the same IP address as the container group and communicates externally through this IP address. • **NAT**: The virtual machine will be assigned an internal IP address but will translate to the container group IP address for external communication. Open ports indicate the exposed ports of the virtual machine, such as the SSH service port 22; not filling in **Open Ports** indicates that all ports are open.
Auxiliary Network Card	Add auxiliary network cards as needed. **Note**: • If auxiliary network card features are required or there are no available types of auxiliary network card networks, please contact the platform administrator for configuration. • SR-IOV types only support Linux operating systems on x86_64 architecture. • Defaults to obtaining IP addresses via DHCP. • After multiple reboots, SR-IOV virtual machines may exhibit two different VFs but with the same MAC address.

8. In the **Initialization Settings** area, refer to the following instructions to configure the relevant information.

Parameter	Description
Keys	Always use SSH keys for remote login verification. This method does not require password validation; it is recommended to log in to the virtual machine using keys. • You can use the keys already available in the platform or create new keys immediately; all relevant keys can be viewed on the **Virtualization**

Parameter	Description
	> Key Pairs page. • Only individuals with the private key can access the virtual machine via SSH. If multiple people are to maintain the virtual machine together, multiple keys can be associated, and private keys can be assigned to different users. If key leakage occurs, the associated key can be promptly revoked to reduce damage. • The public key of the SSH key is stored in the platform in a confidential form; the platform does not store your private key, so please keep it safe by yourself. • Please consult the relevant operating system documentation for the root user password.
Password	Use the operating system user and password for login verification, which can still be updated to the key method later. • The user is only an initial account; after the virtual machine is successfully created, you can also create other operating system users in the virtual machine for login. • The platform encrypts and stores your root user password, and you will not see its plaintext password again, so please keep it safe by yourself.
Start Immediately	Defaults to enabled. Enabling this option will start the virtual machine immediately after creation, otherwise only the virtual machine will be created.

9. (Optional) In the **Advanced Configuration** area, refer to the following instructions to configure the relevant information.

Parameter	Description
Health Check	• **Liveness Check**: Checks if the virtual machine is in a healthy state; if the detection result is abnormal, it will determine whether to restart the instance based on the health check configuration.

Parameter	Description
	• **Availability Check**: Checks if the virtual machine has completed startup and is in a normal service state; if the health status of the virtual machine instance is detected as abnormal, the state of the virtual machine will be updated. For related parameter descriptions.
Node Affinity	• **Preferred**: The virtual machine will be scheduled to nodes that meet affinity requirements whenever possible. The system will determine nodes capable of running the virtual machine by combining affinity weights and other scheduling requirements (e.g., compute resource requirements). • **Required**: The virtual machine will only be scheduled to nodes that fully meet affinity requirements.

10. After confirming that the information is correct, click **Create**.
    Wait for the virtual machine to change from **Creating** to **Running** status.

Related Operations

You can click the ⋮ icon on the right side of the list page or the **Actions** in the upper right corner of the details page to update or delete the virtual machine as needed. For other related operations like resetting passwords or updating keys, please refer to Manage Virtual Machines.

Note:

- Updates can only be performed when the virtual machine is in **Abnormal**, **Unknown**, or **Stopped** status.
- Updates do not support displaying disks that were separately attached or created after the virtual machine was created.
- By default, **Start Immediately** is disabled during updates; you can enable it as needed.

Using the API

A virtual machine is a `VirtualMachine` (`kubevirt.io/v1`). The example below boots from a bootable volume (a `DataSource`), attaches a pod network, and injects an SSH key through cloud-init:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: web-01
  namespace: demo
spec:
  running: false           # set true to start on creation
  dataVolumeTemplates:
    - metadata:
        name: web-01-rootfs
      spec:
        sourceRef:          # clone an existing bootable volume (DataSour
ce)
          kind: DataSource
          name: centos-stream9
          namespace: kube-public
        storage:
          resources:
            requests:
              storage: 20Gi
          storageClassName: rbd-1
          volumeMode: Block
          accessModes:
            - ReadWriteMany
  template:
    metadata:
      labels:
        kubevirt.io/vm: web-01
    spec:
      domain:
        cpu:
          cores: 1
        resources:          # ACP resource-limit model: requests == limit
s
          requests:
            memory: 2Gi
          limits:
            memory: 2Gi
        devices:
          disks:
            - name: rootfs
              disk: { bus: virtio }
            - name: cloudinit
```

```
            disk: { bus: virtio }
        interfaces:
          - name: default
            masquerade: {}
    networks:
      - name: default
        pod: {}
    volumes:
      - name: rootfs
        dataVolume:
          name: web-01-rootfs
      - name: cloudinit
        cloudInitNoCloud:
          userData: |
            #cloud-config
            ssh_authorized_keys:
              - ssh-ed25519 AAAAC3...example user@host
```

```
kubectl apply -f vm.yaml
# Start it (ACP virtual machines use the spec.running field):
kubectl patch vm web-01 -n demo --type merge -p '{"spec":{"running":tru
e}}'
```

The boot disk is provisioned by the `dataVolumeTemplates` entry, which clones the referenced bootable volume. A virtual machine imported with a `spec.instancetype` reference (instead of inline resources) is also supported and is converted to an inline spec when you edit it — see Managing Virtual Machines.

Batch Operations on Virtual Machines

Perform batch operations such as starting, stopping, restarting, and deleting virtual machines.

TOC

Procedure

1. Access the **Container Platform**.
2. Click on **Virtualization** > **Virtual Machines** in the left navigation bar.
3. Locate the target virtual machine, click ⋮ to perform operations on a single virtual machine, or refer to the image below for batch operations on virtual machines.
   Note:
   - The **Start/Batch Start** operation can be executed when the virtual machine is in a paused or stopped state; the **Stop/Batch Stop** operation can be executed when the virtual machine is in a Preparing, Starting, Running, Paused, Unknown, or Exception state; the **Restart/Batch Restart** operation can be executed when the virtual machine is in a Running state.
   - Performing a forced **Restart/Stop** operation on a virtual machine is equivalent to cutting off power to the virtual machine, which may result in loss of data that has not been

written to disk.

4. Complete the operations according to the prompts on the interface. When the virtual machine changes to the states below, the operation is successful.

Operation	Status
Start Virtual Machine	Running
Stop Virtual Machine	Stopped
Restart Virtual Machine	Running

Using the API

ACP virtual machines hold their power state in the `spec.running` field. Start or stop by patching it; use `virtctl` for restart:

```
# Start
kubectl patch vm web-01 -n demo --type merge -p '{"spec":{"running":tru
e}}'    # or: virtctl start web-01 -n demo
# Stop
kubectl patch vm web-01 -n demo --type merge -p '{"spec":{"running":fals
e}}'   # or: virtctl stop web-01 -n demo
# Restart (a running VM)
virtctl restart web-01 -n demo
# Delete
kubectl delete vm web-01 -n demo
```

There is no batch object — apply the same command to each virtual machine in a loop. A forced stop or restart (`virtctl stop --force`) is equivalent to cutting power and may lose data that has not been written to disk.

Pausing and Resuming a Virtual Machine

Pausing a virtual machine freezes its CPU and memory state **without releasing resources or losing data** — the workload is paused until you resume it. Use it to temporarily yield host scheduling or to freeze a virtual machine for inspection and forensics.

Pause and resume act on the virtual machine instance through the KubeVirt `pause` / `unpause` subresources (`subresources.kubevirt.io/v1`); they do not add any CRD.

TOC

Notes

- **Pause** applies to a **Running** virtual machine; **Resume** applies to a **Paused** one.
- Pausing does **not** release CPU/memory and does **not** lose the in-memory state — the guest workload is simply paused.
- Pausing is a disruptive action (the workload is interrupted until you resume), so the console asks for confirmation. Resuming is a safe action and takes effect directly, without confirmation.

- **Permission**: requires the `update` verb on `virtualmachineinstances`.

Pause a virtual machine

Use `virtctl`, which issues an empty-body `PUT` to the virtual machine instance's `pause` subresource:

```
virtctl pause vm web-01 -n demo
```

The underlying request is:

```
PUT /apis/subresources.kubevirt.io/v1/namespaces/demo/virtualmachineinsta
nces/web-01/pause
```

After it succeeds, the virtual machine instance carries the condition `Paused=True` (reason `PausedByUser`) and `VirtualMachine.status.printableStatus` becomes `Paused`. Pausing an already-paused virtual machine returns `409 Conflict`.

Resume a virtual machine

```
virtctl unpause vm web-01 -n demo
```

The underlying request is:

```
PUT /apis/subresources.kubevirt.io/v1/namespaces/demo/virtualmachineinsta
nces/web-01/unpause
```

`VirtualMachine.status.printableStatus` returns to `Running`. Resuming a virtual machine that is not paused returns `409 Conflict`.

Cloning a Virtual Machine

Cloning creates a full copy of an existing virtual machine — including its specification, disks, and network configuration — as a new virtual machine in the **same namespace**. ACP implements this with the KubeVirt `VirtualMachineClone` (`clone.kubevirt.io/v1beta1`) resource, which performs the copy asynchronously through an internal snapshot → restore.

This is different from cloning a **bootable volume** (a PVC or snapshot data source): a VM clone reproduces the whole machine, whereas a bootable-volume clone only duplicates a disk image.

TOC

Prerequisites

- **Snapshot-capable storage.** Because the clone is implemented as an internal snapshot → restore, the source virtual machine's disks must reside on a StorageClass that supports volume snapshots (a matching `VolumeSnapshotClass` exists for that storage backend). If the storage cannot be snapshotted, the clone fails — check the `VirtualMachineClone`

`status.phase` and `status.conditions` for the error. See Clone Virtual Machines on KubeVirt for the full prerequisite list.

Notes

- **Same namespace only.** `VirtualMachineClone.spec.target` has no namespace field, so the new virtual machine is always created in the source's namespace. To copy a virtual machine to another namespace, export and re-import it instead.
- **The new virtual machine appears asynchronously.** Submitting the clone returns immediately; the target is created as the snapshot → restore completes (`status.phase` progresses to `Succeeded`).
- **Cloning a stopped virtual machine is the most reliable**, because the disk state is quiescent.
- **MAC addresses are reset by default** so the clone does not conflict with the source on the network.
- **Kube-OVN networks**: the clone always filters out the source `ovn.kubernetes.io/*` pod annotations so the new virtual machine requests a fresh IP instead of inheriting (and conflicting with) the source IP.
- **Permission**: cloning is gated on the `create` verb for `virtualmachineclones` in the `clone.kubevirt.io` API group. This group is **not** part of the standard **Virtual Machines** function permission, so a user who holds only the Virtual Machines role sees the **Clone** action disabled. A platform or namespace administrator must grant an additional role/binding allowing `create` on `virtualmachineclones.clone.kubevirt.io` before that user can clone.

Clone a virtual machine

Create a `VirtualMachineClone` object in the source namespace. The example below clones `web-01` to `web-01-clone`, resets MAC addresses (`newMacAddresses: {}`), and strips Kube-OVN annotations from the pod template so the clone gets a fresh IP:

```yaml
apiVersion: clone.kubevirt.io/v1beta1
kind: VirtualMachineClone
metadata:
  name: web-01-clone
  namespace: demo
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: web-01
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: web-01-clone
  # Keep all VM-level labels and annotations. Interface MAC addresses are
  # regenerated by default; set newMacAddresses to pin specific MACs inst
ead.
  labelFilters: ["*"]
  annotationFilters: ["*"]
  # Kube-OVN: keep every pod-template annotation EXCEPT the source's
  # ovn.kubernetes.io/* address bindings, so the clone gets a fresh IP in
stead of
  # inheriting the source's. Clone filters are whitelist-based: a bare pa
ttern
  # keeps the matching keys (everything else is dropped) and a "!" prefix
  # excludes — so the "*" entry is required (a lone "!ovn.kubernetes.io/
*" would
  # whitelist nothing and strip ALL pod-template annotations).
  template:
    annotationFilters:
      - "*"
      - "!ovn.kubernetes.io/*"
```

Apply it and watch the clone progress to `Succeeded`:

```shell
kubectl apply -f vmclone.yaml
kubectl get virtualmachineclone web-01-clone -n demo \
  -o jsonpath='{.status.phase}{"\n"}'
```

When `status.phase` is `Succeeded`, the new virtual machine `web-01-clone` exists in the `demo` namespace. `status.restoreName` and `status.snapshotName` reference the intermediate restore and snapshot objects created during the operation.

To copy only a subset of metadata, set the top-level `labelFilters` / `annotationFilters` (they filter the **virtual machine** object's own labels/annotations; `template.*` filters the **pod template** metadata):

```
spec:
  labelFilters:
    - "*"
    - "!some/transient-label"
  annotationFilters:
    - "*"
```

Related operations

After the clone reaches **Running**, manage it like any other virtual machine — see Managing Virtual Machines. Because MAC addresses (and, on Kube-OVN, the IP) are regenerated, verify the new virtual machine's network configuration if your guest pins addresses internally.

Serial Console

The serial console attaches a text terminal (TTY) to a virtual machine's serial port. It is ideal for headless Linux guests, watching kernel boot messages, and capturing kernel panic output — cases where the graphical VNC console is not useful.

The serial console connects to the KubeVirt virtual machine instance `console` subresource over a WebSocket that carries the raw TTY byte stream (subprotocol `plain.kubevirt.io`).

TOC

Notes

- The serial console is available whenever the virtual machine has a **running instance** (Running, Paused, Starting, or Stopping). It is unavailable for a stopped virtual machine, because no instance exists — the endpoint returns `404`.
- The guest must expose a serial getty on its console — Linux cloud images typically set `console=ttyS0` on the kernel command line. Windows guests have no serial console; use VNC instead.
- **Permission**: requires the `get` verb on `virtualmachineinstances/console`.

Open the serial console

```
virtctl console web-01 -n demo
```

This upgrades to a WebSocket on the virtual machine instance `console` subresource:

```
GET /apis/subresources.kubevirt.io/v1/namespaces/demo/virtualmachineinsta
nces/web-01/console
Upgrade: websocket
Sec-WebSocket-Protocol: plain.kubevirt.io
```

A running instance returns `101 Switching Protocols` and then streams the raw TTY bytes; a stopped virtual machine returns `404` (no instance).

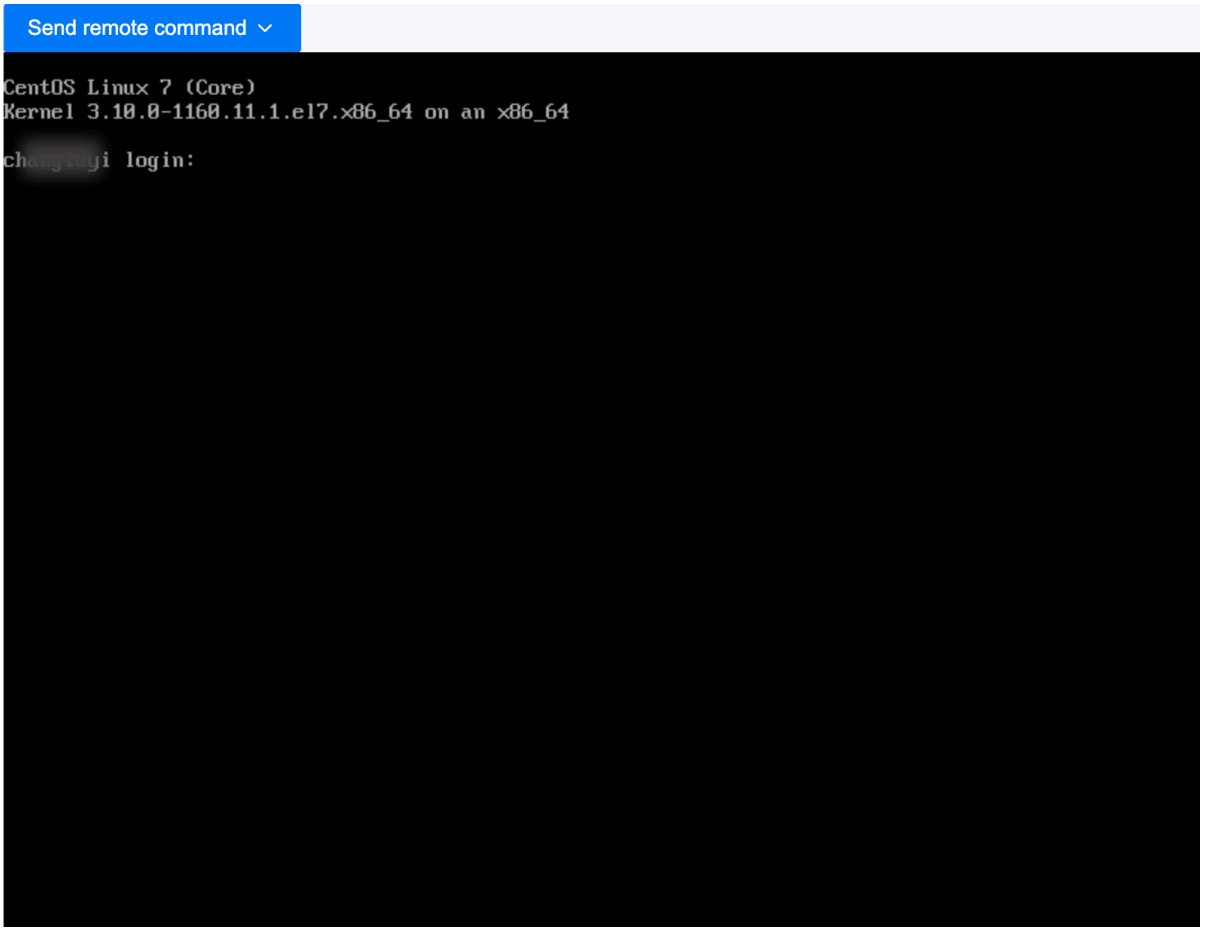
Logging into the Virtual Machine using VNC

Log into the virtual machine using the Web Console (VNC) as an emergency operation method.

TOC

Procedure

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click ⋮ > **VNC Login**.
4. The console window will open automatically; you will need to enter your username and password to log in.



Note:

- Supports sending common keyboard commands.
- Supports copying and pasting commands and parameters.

Using the API

```
virtctl vnc web-01 -n demo
```

`virtctl vnc` launches a local VNC viewer connected to the virtual machine instance `vnc` subresource over a WebSocket:

```
GET /apis/subresources.kubevirt.io/v1/namespaces/demo/virtualmachineinsta
nces/web-01/vnc
Upgrade: websocket
```

A running instance returns `101 Switching Protocols`; the connection requires the `get` verb on `virtualmachineinstances/vnc`.

Managing Key Pairs

Create, update or delete key pairs.

> A key pair is stored as a Kubernetes `Secret` of type `kubernetes.io/ssh-public-key`; only the **public** key is kept — the platform never stores your private key. Key pairs are consumed when creating a virtual machine through **Associated SSH Keys** (injected via cloud-init).

TOC

Creating Key Pairs

1. Navigate to **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Key Pairs**.
3. Click **Create Key Pair**.
   Currently, only SSH type key pairs are supported. You can manually import keys or let the system automatically generate a key pair. When using the system-generated key pair, the

platform supports automatically downloading the private key to your local machine. The platform will not save the private key.

4. Click **Create**.

Updating Key Pairs

1. Navigate to **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Key Pairs**.
3. Locate the ***Key Pair Name***, click ⋮ > **Update**.
4. After re-importing or having the system generate a new key pair, click **Update**.

Deleting Key Pairs

1. Navigate to **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Key Pairs**.
3. Locate the ***Key Pair Name***, click ⋮ > **Delete**, and confirm.

Using the API

A key pair is a `Secret` of type `kubernetes.io/ssh-public-key` whose `ssh-publickey` value is the public key:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-key
  namespace: demo
type: kubernetes.io/ssh-public-key
stringData:
  ssh-publickey: "ssh-ed25519 AAAA...example user@host"
```

```
kubectl apply -f keypair.yaml
kubectl get secret my-key -n demo -o jsonpath='{.type}{"\n"}'   # kuberne
tes.io/ssh-public-key
kubectl delete secret my-key -n demo
```

Reference the key when creating a virtual machine through **Associated SSH Keys**; the platform injects the public key into the guest via cloud-init `ssh_authorized_keys`.

Managing Virtual Machines

The operations below are available from the virtual machine **list** (row ⋮ menu) and the **detail** page. The following operations are documented as API/CLI references: Clone, Pause and Resume, and Serial Console. Each operation also has an equivalent API call — see **Using the API** below.

TOC

Reset Password

Reset the **root** user password. This password also serves as the login password for the virtual machine when logging in using a password.

Procedure

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the virtual machine and select ⋮ > **Reset Password**.
4. Set the password.
5. Click **Reset**.
   Note: Please keep your password safe. To ensure environment security, the platform encrypts and stores your password, and you will not be able to see the plaintext password again.

Update Key

Update the SSH keys.

Procedure

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the virtual machine and select ⋮ > **Update Key**.
4. Select one or more associated keys, or **Create Key**.
5. Choose whether to restart immediately; updating keys requires a restart of the virtual machine to take effect.
6. Click **Update**.

Update Specifications

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the target virtual machine and click ⋮ > **Update Specifications**.
4. Modify the relevant resources based on the platform's recommended scenarios or custom needs.
5. Choose whether to **Restart Immediately**; the configuration will take effect after restarting.
6. Click **Update**.

Live Migration

Note: For prerequisites, the cluster-wide migration limits and per-namespace policies, and the API, see Virtual Machine Live Migration.

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the target virtual machine and click ⋮ > **Live Migration**.
4. Click **Confirm**.

Update NAT Network Configuration

When using NAT network mode, the platform by default opens port 22 for SSH services, and you can open other ports as needed.

Procedure

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click ***Virtual Machine Name***.
4. In the **Basic Information** section, click the icon to the right of **Open Port**.

5. Enter the port number and press the Enter key to confirm.
6. Choose whether to **Restart Immediately**; the configuration will take effect after restarting.
7. Click **Update**.

Update Tags and Annotations

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click ***Virtual Machine Name***.
4. In the **Basic Information** section, click the icon to the right of **Tags** or **Annotations**.
5. Configure as needed and click **Update**.

Add Service

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click ***Virtual Machine Name***.
4. In the **Login Information** section, click the icon to the right of Internal Route.
5. Refer to the Create Service page for quick addition of internal routes for the virtual machine.
6. Click **Confirm**.

Reinstall Operating System

It is strongly recommended to back up data before reinstalling the operating system to prevent data loss.

Note: This operation will clear all data in the virtual machine's **system disk**, as well as all **snapshots**, and is irreversible. Please proceed with caution!

Procedure

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the virtual machine and select ⋮ > **Reinstall Operating System**.
4. In the **Reinstall Operating System** window, configure the following parameters.
   - **Provisioning Method**: Currently supports public images.
   - **Select Image**: By default, the current operating system image will be used for reinstallation. If you wish to reinstall a new operating system, first select the operating system of the virtual machine image, then choose the virtual machine image that belongs to that operating system.
5. Click **Reinstall**.

Configure IP

Assign an IP to the virtual machine using dynamic allocation (DHCP) or bind a fixed IP to the virtual machine; the new IP will take effect after the virtual machine is restarted.

Procedure

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the target virtual machine and click ⋮ > **Configure IP**.
4. Configure the **IP Address**.
   - Fill in an available IP: binding a fixed IP means that even after restarting, the virtual machine will consistently use this IP address.
   - Leave the option blank: this will use dynamic allocation (DHCP) to acquire an IP, assigning the IP when the virtual machine starts and releasing it when the virtual machine stops.
5. Choose whether to **Restart Immediately**; the configuration will take effect after restarting.

6. Click **Configure**.

Using the API

The console operations map to standard calls on the `VirtualMachine` resource:

- **Update Specifications** — patch the inline resources (ACP uses a resource-limit model, so set `requests` and `limits` to the same values); restart the virtual machine for the change to take effect:

```
kubectl patch vm web-01 -n demo --type merge -p \
  '{"spec":{"template":{"spec":{"domain":{"resources":{"requests":{"cp
u":"2","memory":"4Gi"},"limits":{"cpu":"2","memory":"4Gi"}}}}}}}'
```

- **Update Tags and Annotations**:

```
kubectl label vm web-01 -n demo app=billing
kubectl annotate vm web-01 -n demo cpaas.io/maintainer=kim
```

- **Live Migration** — create a `VirtualMachineInstanceMigration`; see Virtual Machine Live Migration.

Imported Virtual Machines Using an InstanceType Spec

A virtual machine imported from another platform (for example via Forklift) may define its CPU and memory through a referenced `VirtualMachineClusterInstancetype` (`spec.instancetype`) instead of inline `domain.resources`. The referenced instance type is resolved so that its effective CPU and memory can be displayed.

When you **edit** such a virtual machine's specification, it is converted **once** to a custom inline spec — `domain.resources` is written and `spec.instancetype` is removed — and the change takes effect after a restart. A raw imported virtual machine without an ACP label can only be edited as YAML.

```
# List the cluster instance types a VM can reference:
kubectl get virtualmachineclusterinstancetype
```

Monitoring and Alerts

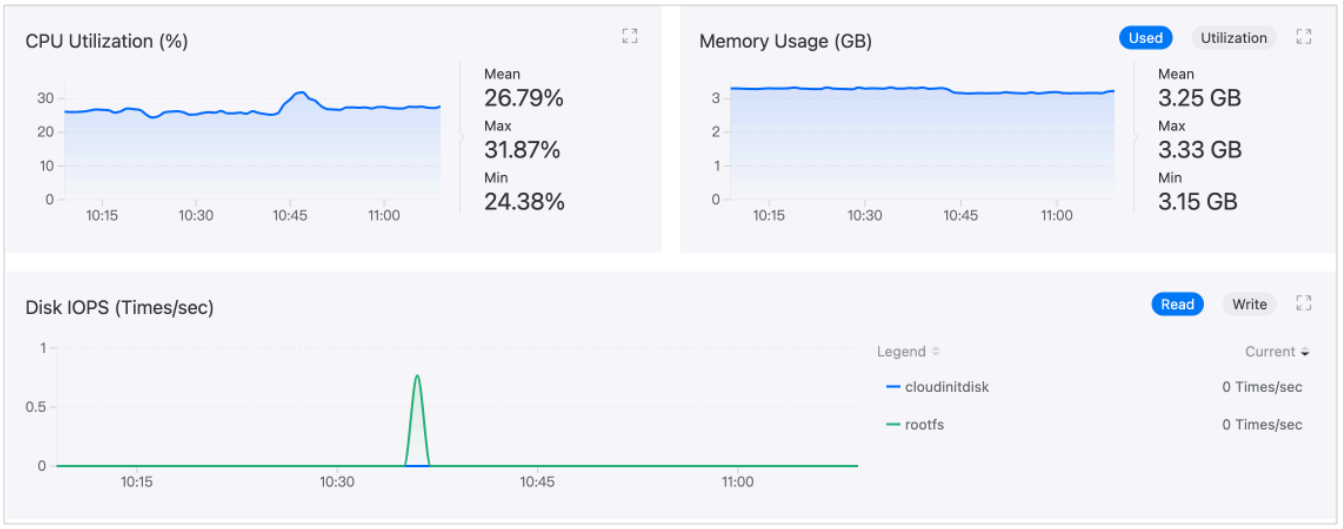
Monitor and alert on virtual machines in terms of CPU, memory, storage, and network. To facilitate timely alerts, notification policies can also be configured.

The intuitively presented monitoring data can be used to provide decision-making support for operations inspection or performance tuning, while the comprehensive alerting and notification mechanism will help ensure the stable operation of virtual machines.

TOC

Monitoring

By default, the platform collects commonly used performance monitoring metrics for virtual machines, including CPU, memory, storage, and network. Navigate to **Virtualization** > **Virtual Machines**, and on the **Monitoring** tab in the virtual machine details, you can view real-time monitoring data for the metrics.



Alerts

Configuring Alert Policies

To enable alerts, you must first create an alert policy. An alert policy describes the objects you wish to monitor, the conditions under which you wish to be alerted, and how you will be notified of relevant alerts. Navigate to **Container Platform** > **Virtualization** > **Virtual Machines**, and in the virtual machine details, click **Create Alert Policy** on the **Alerts** tab to complete the configuration.

Parameter	Description
Alert Type	- Metric Alert: The monitored object is a platform predefined metric, such as *Memory Usage Rate*. - Event Alert: The monitored object is the cause of an event, that is, the reason the virtual machine transitioned to its current state, e.g., BackOff, Pulling, Failed.
Trigger Condition	Composed of comparison operators, alert thresholds, and duration. By comparing the real-time monitoring results with the set thresholds, it determines whether to alert. If a duration is set, the platform will also compare the duration for which the monitored object has been in the alert state.
Alert Level	- Hint: The monitored object has expected issues that do not immediately affect business operations but pose potential risks. For example, if CPU usage exceeds 70% for 3 minutes.

Parameter	Description
	- Warning: The monitored object has operational risks that may affect normal business operations if not addressed promptly. For example, if CPU usage exceeds 80% for 3 minutes. - Serious: The monitored object has known issues that may lead to platform functionality failures, affecting normal business operations. - Disaster: The monitored object has failed, resulting in platform service interruptions, data loss, with significant impact.

Tip: The virtual machine alerting function is similar to the platform's general alerting function. For more detailed configuration guidance, refer to the platform's general alerting documentation.

Handling Alerts

Navigate to the **Alerts** tab, and if there are alert status strategies indicated, please address them promptly.

Binding Notification Policies

In addition to real-time alerts on the **Alerts** tab, the platform also supports sending alert information via email, SMS, and other means to relevant personnel, notifying them to take necessary measures to resolve issues or prevent failures. The notification policy needs to be set up by contacting the administrator.

Using the API

Alert policies are backed by Prometheus rules. You can also define an alert directly as a `PrometheusRule` (`monitoring.coreos.com/v1`):

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: vm-high-memory
  namespace: demo
spec:
  groups:
    - name: vm.rules
      rules:
        - alert: VMHighMemoryUsage
          expr: kubevirt_vmi_memory_used_bytes / kubevirt_vmi_memory_avai
lable_bytes > 0.8
          for: 3m
          labels:
            severity: warning
          annotations:
            summary: "VMI {{ $labels.name }} memory usage above 80%"
```

KubeVirt emits per-virtual-machine-instance metrics (prefix `kubevirt_vmi_*`, with `namespace` and `name` labels) that you can query in Prometheus or use in your own dashboards — for example `kubevirt_vmi_memory_used_bytes`, `kubevirt_vmi_network_receive_bytes_total`, `kubevirt_vmi_storage_read_traffic_bytes_total`, and `kubevirt_vmi_migration_data_processed_bytes`. Exact metric names vary by KubeVirt version; list the full set with the query `{__name__=~"kubevirt_vmi_.+"}`.

Quick Location of Virtual Machines

The platform supports displaying the list of virtual machines by cluster, allowing platform administrators to quickly locate the namespace of the virtual machine and perform operations such as scaling up or troubleshooting, thereby improving operational efficiency.

TOC

Prerequisites

Ensure that the virtualization feature is enabled for the current cluster before use. Please refer to Install.

Procedure

1. Go to **Administrator**.
2. In the left navigation bar, click **Virtualization Management** > **Virtual Machines**.
3. Select ***Cluster*** to view the list of virtual machines in that cluster.

4. You can quickly locate the virtual machine by its name, IP address, or creator.
5. Click on the virtual machine ***Name*** link to enter the details page of that virtual machine, where you can perform operations such as scaling up or troubleshooting.

Using the API

List and locate virtual machines across all namespaces from the command line:

```
# All virtual machines in the cluster, with status:
kubectl get vm -A -o wide

# A specific virtual machine by name across namespaces:
kubectl get vm -A --field-selector metadata.name=web-01

# Virtual machine instances show the assigned IP address and node:
kubectl get vmi -A -o wide
```

How To

Using the virtctl Command-Line

Consoles and access

Uploading and exporting images

Hot-plugging disks

Power and migration

Editing and sealing images with libguestfs

Configuring USB host passthro

Feature Overview

Use Cases

Prerequisites

Steps

Operation Result

Learn More

Installing G

QEMU guest ag

VirtIO drivers (V

Virtual TPM (vT

Virtual Mach

CPU and Memory Hotplug

How It Works

Prerequisites

Step 1: Create a VM with Hotplug Headroom

Step 2: Hotplug CPU

Step 3: Hotplug Memory

Scaling Down: What Works Live and What

Verified Behavior Summary

Troubleshooting

Virtual Machine Recovery

Steps to Operate

ettin

Clone Virtual Machines on KubeVirt

Ensure Prerequisites

Start Quickly

Understand the VirtualMachineClone Obje

Migrate Virt

Prerequisites

Prepare the Mig

Cold Migration

chi

Create a VM Template from an Existing Virtual Machine

Using the virtctl Command-Line Tool

`virtctl` is the standard KubeVirt command-line client. It complements `kubectl` with commands that need a streaming connection or a subresource call — opening consoles, uploading and exporting images, hot-plugging disks, and triggering migrations. It ships with KubeVirt; download the binary that matches the cluster's KubeVirt version from the KubeVirt releases and place it on your `PATH`.

```
# Example: download virtctl for the cluster's KubeVirt version
VERSION=$(kubectl get kubevirt kubevirt -n kubevirt -o jsonpath='{.statu
s.observedKubeVirtVersion}')
curl -L -o virtctl "https://github.com/kubevirt/kubevirt/releases/downloa
d/${VERSION}/virtctl-${VERSION}-linux-amd64"
chmod +x virtctl && sudo mv virtctl /usr/local/bin/
```

TOC

Consoles and access

```
virtctl console web-01 -n demo        # serial (text) console — see Seria
l Console
virtctl vnc web-01 -n demo            # graphical VNC console — see VNC
virtctl ssh user@web-01 -n demo       # SSH into the VM (uses the VM's po
d network)
virtctl scp ./file user@web-01:/tmp -n demo
```

See Serial Console and Logging into the Virtual Machine using VNC.

Uploading and exporting images

```
# Upload a local image into a DataVolume (CDI cdi-uploadproxy must be rea
chable):
virtctl image-upload dv my-image --size=20Gi --storage-class=rbd-1 \
  --image-path=./my-image.qcow2 -n demo

# Export a VM disk or snapshot for download:
virtctl vmexport download my-export --vm=web-01 --output=disk.img.gz -n d
emo
```

See Uploading a Local Image and Creating a Virtual Machine for the full local-image upload and virtual-machine creation workflow, and Exporting Virtual Machine Images.

Hot-plugging disks

```
virtctl addvolume web-01 --volume-name=datadisk-1 --persist -n demo
virtctl removevolume web-01 --volume-name=datadisk-1 -n demo
```

See Managing Virtual Disks.

Power and migration

```
virtctl start web-01 -n demo
virtctl stop web-01 -n demo
virtctl restart web-01 -n demo
virtctl pause vm web-01 -n demo        # freeze CPU/memory
virtctl unpause vm web-01 -n demo
virtctl migrate web-01 -n demo         # start a live migration
virtctl migrate-cancel web-01 -n demo  # cancel an in-progress live migra
tion
```

See Virtual Machine Live Migration and Pausing and Resuming a Virtual Machine.

Editing and sealing images with libguestfs

`virtctl guestfs` starts an interactive pod with the libguestfs tools attached to a PVC, so you can inspect or modify a disk image offline — for example to reset a machine ID or generalize a template before publishing it as a bootable volume:

```
virtctl guestfs my-pvc -n demo
# inside the pod:
virt-customize -a /dev/vda --root-password password:'...'
virt-sysprep -a /dev/vda        # generalize (remove host-specific data)
before templating
```

Configuring USB host passthrough

TOC

Feature Overview

USB(Universal Serial Bus) pass-through feature enables you to access and manage USB devices from a virtual machine.

Use Cases

Some applications running in virtual machines (VMs) have encryption requirements and need to interact with dedicated USB devices. In such cases, it is necessary to passthrough the USB devices from the host machine to the VM.

Prerequisites

- The platform version must be at least v3.18.

Steps

1 Expose USB devices

To assign a USB device to a VM, the USB device must be exposed via a **ResourceName**. This can be configured by editing the `spec.permittedHostDevices.usbHostDevices` section in the **HyperConverged CR** under the kubevirt namespace.

Below is an example configuration for a USB device with **ResourceName** kubevirt.io/storage, where the vendor is `0bda` and the product is `8812`:

```
spec:
  permittedHostDevices:
    usbHostDevices:
      - resourceName: kubevirt.io/storage
        selectors:
          - vendor: '0bda'
            product: '8812'
```

Tip

The vendor and product identifiers of a USB device can be obtained using the `lsusb` command. For example:

```
lsusb
Bus 001 Device 007: ID 0bda:8812 Realtek Semiconductor Corp. RT
L8812AU 802.11a/b/g/n/ac 2T2R DB WLAN Adapter
```

This command lists all connected USB devices, where ID displays the vendor:product pair.

2 Assign USB devices to a Virtual Machine

Now, in the VM configuration, you can add `spec.template.spec.domain.devices.hostDevices.deviceName` to reference the `ResourceName` provided in the previous step and assign it a local name. For example:

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          hostDevices:
            - deviceName: kubevirt.io/storage
              name: usb-storage
```

Tip

Ensure the VM is stopped before editing the configuration.

Operation Result

After completing the configuration, execute the `lsusb` command within the virtual machine. If the output lists the host node's USB device, the passthrough was successful. For example:

```
lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 0bda:8812 Realtek Semiconductor Corp. RTL8812AU 80
2.11a/b/g/n/ac 2T2R DB WLAN Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Learn More

You may want to passthrough multiple USB devices to a virtual machine, such as a keyboard, mouse, or smart card device. We support assigning multiple USB devices under the same resourceName. Here's how to configure it:

1 Expose multiple USB devices

```yaml
spec:
  permittedHostDevices:
    usbHostDevices:
      - resourceName: kubevirt.io/peripherals
        selectors:
          - vendor: '0bda'
            product: '8812'
          - vendor: '062a'
            product: '4102'
          - vendor: '072f'
            product: 'b100'
```

> **Tip**
>
> Note: All USB devices must be physically connected and detected on the host to ensure successful assignment to the virtual machine.

2 Assign USB devices to a Virtual Machine

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          hostDevices:
            - deviceName: kubevirt.io/peripherals
              name: local-peripherals
```

Installing Guest Tools (Guest Agent, VirtIO Drivers, vTPM)

Installing the right tools inside the guest unlocks features the platform relies on — reporting the IP address, graceful shutdown, application-consistent snapshots, and (for Windows) usable disk and network drivers.

TOC

QEMU guest agent

The QEMU guest agent lets the platform read the guest IP, shut the guest down gracefully, and freeze the filesystem for **application-consistent** snapshots. Without it, online snapshots are only crash-consistent.

- **Linux** — install and enable the agent from the distribution packages:

```
# RHEL / CentOS / Fedora
sudo dnf install -y qemu-guest-agent && sudo systemctl enable --now qemu-guest-agent
# Debian / Ubuntu
sudo apt-get install -y qemu-guest-agent && sudo systemctl enable --now qemu-guest-agent
```

- **Windows** — install the guest agent from the virtio-win media (`guest-agent\qemu-ga-x86_64.msi`).

When the agent is running, `VirtualMachineInstance.status.conditions` includes `AgentConnected=True` and the IP address appears in the console and in `status.interfaces` .

VirtIO drivers (Windows)

Windows has no built-in VirtIO drivers, so a freshly installed Windows guest cannot see its VirtIO disk or NIC until the drivers are installed. Attach the **virtio-win** container disk as a CD-ROM and install the drivers:

- During Windows setup, **Load driver** from the virtio-win CD-ROM (`viostor` for the boot disk).
- After installation, run `virtio-win-guest-tools.exe` from the same media to install all drivers plus the guest agent.

Virtual TPM (vTPM)

Windows 11 and BitLocker require a TPM 2.0 device. Add a virtual TPM to the virtual machine through `spec.template.spec.domain.devices.tpm` :

```
spec:
  template:
    spec:
      domain:
        devices:
          tpm: {}
        firmware:
          bootloader:
            efi:
              secureBoot: true   # Windows 11 also requires UEFI + Secure
Boot
```

Note that a vTPM's persistent state is **not** included in virtual machine snapshots or clones.

Virtual Machine Live Migration

TOC

Overview

The virtual machine live migration technology allows for moving a virtual machine from one physical server to another without shutting down or interrupting the virtual machine. The

platform's virtual machine solution is implemented based on the open-source component KubeVirt, which uses **pre-copy** memory migration by default.

Pre-Copy

Pre-copy memory migration is a commonly used virtual machine migration technology that ensures service continuity during migration by pre-copying the virtual machine's memory data. The specific process is as follows:

1. **Initial Phase**: At the start of the migration, the source host will copy the virtual machine's memory pages to the target host while the virtual machine continues to run. Because the virtual machine continues running, some memory pages may be modified during the copying process.
2. **Iterative Copying**: The source host repeatedly copies the modified memory pages to the target host until the number of modified pages decreases to an acceptable level. Each round of copying is called an iteration, and the number of unmodified memory pages gradually decreases after each iteration.
3. **Stop and Copy**: When the remaining un-copied memory pages are sufficiently few, the virtual machine will pause briefly (usually only a few seconds to a dozen seconds), during which the last memory pages are copied to the target host, and the virtual machine's CPU and device states are synchronized to the target host.
4. **Resume Operation**: The virtual machine resumes operation on the target host.

Constraints and Limitations

It is recommended that the two physical machines involved in the live migration operation use the same hardware configuration. If the configurations are inconsistent (for example, different CPU models), migration may fail.

Prerequisites

Please enable the relevant virtual machine live migration functions in advance.

Operation Steps

Deploy kubevirt-operator

Note: For detailed steps and parameter explanations, please refer to Install.

1. Go to **Administrator**.
2. In the left navigation bar, click **App Store Management** > **Operators**.
3. Click **Cluster** at the top of the page to switch to the cluster where the Operator needs to be deployed.
4. In the OperatorHub tab, click **Deploy** on the **KubeVirt HyperConverged Cluster Operator** card.
5. Configure the parameters as needed and click **Deploy**. You can check the Operator deployment status in the **Deployed** tab.

Create HyperConverged Instance

For specific creation steps, please refer to Create HyperConverged Instance.

Prepare the Virtual Machine

Note: It is recommended to use the Kube-OVN Underlay network. For related configurations, please refer to Create Subnet (Kube-OVN Underlay Network).

1. Go to **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machine**.
3. Click **Create Virtual Machine**.
4. Click **More** in the **Basic Information** area to expand more configuration options, and click **Add** corresponding to **Annotations**, adding annotations according to the key-value pairs below. If the network plugin is Kube-OVN, there is no need to manually fill in this annotation.
   Note: Due to form restrictions, please enter the **value** of the annotation first before entering the **key** of the annotation.

	Annotation
Value	true
Key	kubevirt.io/allow-pod-bridge-network-live-migration

5. Configure other virtual machine parameters as needed. For specific parameter descriptions, please refer to the relevant product documentation.

Parameter	Description
Volume Mode	Must use **Block Mode**.
Storage Class	Must use **CephRBD block storage type storage class**.
Network Mode	Recommended to use **Bridge**.

6. Click **Create**.

Start Live Migration

Note: Live migration can only be started when the virtual machine status is **Running**.

1. Go to **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machine**.
3. Start the live migration. There are two ways to do this:
   - Click ⋮ > **Live Migration** on the right side of the virtual machine that needs to be migrated in the list.
   - Click the name of the virtual machine that needs to be migrated in the list to enter the detail information page, then click **Actions** > **Live Migration**.
4. Click **Confirm**. You can check the migration progress through **Virtual Machine Status** or **Real-Time Events**. When the status changes from **Migrating** to **Running**, or when a real-time event appears with information like **Migrated: The VirtualMachineInstance migrated to node 10.1.1.1.**, it indicates that the migration was successful.

Using the API

Start a live migration by creating a `VirtualMachineInstanceMigration` that names the running virtual machine instance:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: migrate-web-01
  namespace: demo
spec:
  vmiName: web-01
```

```
kubectl apply -f migration.yaml
# phase progresses Pending -> Scheduling -> Running -> Succeeded
kubectl get virtualmachineinstancemigration migrate-web-01 -n demo -o jso
npath='{.status.phase}{"\n"}'
```

When the phase is `Succeeded`, the instance is running on the target node — see `VirtualMachineInstance.status.migrationState.targetNode`. **To cancel an in-progress migration, delete its `VirtualMachineInstanceMigration` object** (there is no cancel field):

```
kubectl delete virtualmachineinstancemigration migrate-web-01 -n demo
```

Migration Settings

Cluster-wide live-migration limits and per-namespace overrides are configured through the `HyperConverged` resource and cluster-scoped `MigrationPolicy` objects, as described below.

Cluster defaults

The cluster defaults are the KubeVirt `liveMigrationConfig`, which is owned by the HyperConverged operator. Set it on the `HyperConverged` resource — **never** directly on `KubeVirt`, because the operator reconciles that value back:

```
kubectl get hyperconverged kubevirt-hyperconverged -n kubevirt -o jsonpat
h='{.spec.liveMigrationConfig}'
# {"parallelMigrationsPerCluster":5,"parallelOutboundMigrationsPerNode":
2,
#  "completionTimeoutPerGiB":150,"progressTimeout":150,
#  "allowAutoConverge":false,"allowPostCopy":false}
```

Field	Default	Meaning
`parallelMigrationsPerCluster`	5	Maximum concurrent migrations cluster-wide
`parallelOutboundMigrationsPerNode`	2	Maximum concurrent outbound migrations per node
`completionTimeoutPerGiB`	150	Seconds per GiB before the migration is aborted
`progressTimeout`	150	Seconds without progress before aborting
`allowAutoConverge`	false	Throttle the guest CPU to help the migration converge
`allowPostCopy`	false	Switch to post-copy if pre-copy does not converge

Namespace policies

A cluster-scoped `MigrationPolicy` overrides the defaults for the namespaces (or virtual machines) matched by its selector:

```yaml
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
  name: policy-fast
spec:
  selectors:
    namespaceSelector:
      kubernetes.io/metadata.name: demo
  allowAutoConverge: true
  completionTimeoutPerGiB: 100
```

A virtual machine whose namespace matches no policy uses the cluster defaults.

CPU and Memory Hotplug

This document describes how to add CPU and memory to a running virtual machine (VM) without shutting it down, and — equally important — how different guest operating systems react to the hotplug. The platform side is identical for every OS, but whether the guest actually *uses* the new resources depends on the guest kernel or driver. The per-OS results marked *verified* below were tested on a live cluster with CentOS 7.9, Ubuntu 20.04, Debian 13, and Windows Server 2022 guests; compatibility notes about other releases (Debian 12, Ubuntu 22.04/24.04, RHEL 8.10/9.4, other Windows editions) are based on upstream and vendor documentation, not on those tests.

TOC

How It Works

The platform's virtualization is based on KubeVirt, which applies CPU and memory changes to running VMs through the `LiveUpdate` rollout strategy (the platform default):

- **CPU hotplug** adds vCPUs by increasing the number of CPU **sockets**. The hotplug ceiling is `maxSockets`; the extra sockets are pre-wired as empty slots that QEMU can populate later.
- **Memory hotplug** grows guest memory through a **virtio-mem** device. The ceiling is `maxGuest`; the delta between `guest` and `maxGuest` is backed by a virtio-mem device that can be resized at runtime — in both directions, as long as the size stays at or above the boot-time memory (scaling down is best-effort, see below).
- Both changes are delivered by an **automatic live migration**: after you edit the VM spec, the platform migrates the VM to a new `virt-launcher` pod whose resource allocation matches the new size. You do not trigger the migration yourself — just be aware that each hotplug produces one migration, and that a VM which cannot live-migrate cannot be hotplugged.

Progress is reported on the VirtualMachineInstance (VMI):

- Condition `HotVCPUChange` / `HotMemoryChange` — a hotplug is in flight.
- `status.currentCPUTopology` — the CPU topology the domain currently has.
- `status.memory` — `guestAtBoot` / `guestCurrent` / `guestRequested`; a pending memory hotplug shows `guestRequested` > `guestCurrent`.
- Condition `RestartRequired` on the VM — the requested change cannot be applied live (for example, a CPU socket reduction, or memory below the boot-time value).

Prerequisites

- The VM must be **live-migratable**: the `LiveMigratable` condition on the VMI must be `True`. Hotplug rides on live migration; a non-migratable VM cannot be resized online. Storage-wise this means every **PVC/DataVolume-backed** disk needs shared storage with `ReadWriteMany` access mode — including CD-ROMs backed by an RWO PVC. Volumes that are not PVC-backed (`containerDisk`, cloud-init, `emptyDisk`) do not need RWX.

Non-storage factors (host devices, certain CPU configurations) can also make a VM non-migratable; note the condition message reports only the *first* blocking reason it finds.

- The VM must have hotplug headroom (`maxSockets` / `maxGuest`). The effective ceilings are fixed when the VMI starts — either from explicit values in the VM spec, or auto-computed by the platform when omitted (see below). They cannot be raised on a running VM: growing beyond the current ceiling requires a restart.
- For **memory** hotplug, the guest must have a virtio-mem driver that can negotiate the feature set the platform's QEMU offers — in particular the `UNPLUGGED_INACCESSIBLE` feature. For Linux guests this is a kernel capability (see the per-OS results below); for Windows guests it is the `viomem` driver from the virtio-win package, which is **not installed by default** on existing VMs (see the Windows row below). Check inside a Linux guest after a hotplug attempt:

```
dmesg | grep virtio_mem
# healthy:  virtio_mem virtio1: start address: ..., region size: ...
# too old:  virtio_mem virtio1: virtio: device refuses features: 3
```

Step 1: Create a VM with Hotplug Headroom

The relevant fields in the VM template:

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      domain:
        cpu:
          sockets: 1        # current vCPU count = sockets × cores × thre
ads
          cores: 1
          threads: 1
          maxSockets: 4     # hotplug ceiling: up to 4 sockets without re
start
        memory:
          guest: 2Gi        # current guest memory
          maxGuest: 8Gi     # hotplug ceiling: up to 8Gi without restart
```

If `maxSockets` / `maxGuest` are omitted, the platform computes them automatically when the VMI starts (by default 4× the boot value; the ratio and absolute caps are configurable cluster-wide via `liveUpdateConfiguration`). The computed values land on the VMI, not on the VM spec — a VM created through the web console, whose template only carries `resources.requests` / `resources.limits`, still gets full hotplug headroom. Setting the maximums explicitly is still recommended so the ceiling is a conscious choice.

Inside the guest this headroom is already visible at boot: a VM created with the spec above shows 1 CPU online and CPUs 1–3 as offline slots, and 2Gi of memory.

WARNING

For a VM whose template expresses its size only through `resources.requests` / `resources.limits` — which is what the web console generates — the *first* resize behaves differently per resource (verified against such a VM):

- **Memory**: adding `domain.memory.guest` to the running VM is treated as a live update. The hotplug proceeds normally — no restart needed.
- **CPU**: the template has no `cpu` block, and *adding one* (even just `cpu.sockets`) is a non-live-updatable change. The VM gets `RestartRequired` (`a non-live-updatable field was`

`changed in the template spec`); the first CPU resize therefore needs one restart, after which the spec carries an explicit `cpu` block and every subsequent socket increase is applied live.

- **Do not combine them**: if one patch adds both the `cpu` block and `memory.guest`, the CPU change sets `RestartRequired` and that condition also suppresses the (otherwise live-applicable) memory hotplug. Patch memory separately if you need it applied online.

Step 2: Hotplug CPU

Increase the socket count on the running VM:

```
kubectl patch vm <vm-name> -n <namespace> --type=merge \
  -p '{"spec":{"template":{"spec":{"domain":{"cpu":{"sockets":2}}}}}}'
```

What happens next (typically completes within ~30 seconds):

1. The VMI gets the `HotVCPUChange` condition.
2. The platform automatically live-migrates the VM to a new virt-launcher pod.
3. `status.currentCPUTopology` reports the new socket count and QEMU hot-adds the vCPU.

Watch it:

```
kubectl get vmi <vm-name> -n <namespace> \
  -o jsonpath='{.status.currentCPUTopology}'
```

Guest OS Behavior — CPU

The vCPU is hot-added at the hypervisor level for every OS. Whether it is **used** immediately depends on the guest:

Guest OS	Behavior (verified)
CentOS 7.9	New vCPU is onlined **automatically**. RHEL-family ships `40-redhat.rules`, which onlines any hot-added CPU unconditionally. `nproc` reflects the new count within seconds of the migration finishing.
Ubuntu 20.04	New vCPU appears but stays **offline** (`nproc` unchanged). Ubuntu's `40-vm-hotadd.rules` only auto-onlines CPUs on Hyper-V and Xen — it does not match KVM/QEMU guests.
Debian 13	Same as Ubuntu — the vCPU is added but stays offline; Debian ships no auto-online rule at all.
Windows Server 2022 (Standard)	New vCPUs are picked up **automatically and immediately** — `%NUMBER_OF_PROCESSORS%` reflected the new count in a live session with no reboot and no manual step.

To bring the CPU online on Ubuntu/Debian (and any other distribution without a KVM-covering udev rule), either online it manually:

```
echo 1 > /sys/devices/system/cpu/cpu1/online
```

or install a udev rule (this is exactly what RHEL ships) so future hotplugs are picked up automatically:

```
cat <<'EOF' > /etc/udev/rules.d/99-kvm-cpu-hotplug.rules
SUBSYSTEM=="cpu", ACTION=="add", TEST=="online", ATTR{online}=="0", ATTR
{online}="1"
EOF
udevadm control --reload
```

Step 3: Hotplug Memory

Increase guest memory on the running VM:

```
kubectl patch vm <vm-name> -n <namespace> --type=merge \
  -p '{"spec":{"template":{"spec":{"domain":{"memory":{"guest":"4G
i"}}}}}}'
```

The platform live-migrates the VM and resizes the virtio-mem device. Track the result:

```
kubectl get vmi <vm-name> -n <namespace> -o jsonpath='{.status.memory}'
# {"guestAtBoot":"2Gi","guestCurrent":"4Gi","guestRequested":"4Gi"}   <-
success
# {"guestAtBoot":"2Gi","guestCurrent":"2Gi","guestRequested":"4Gi"}   <-
pending: guest cannot take it
```

Guest OS Behavior — Memory

This is where guest driver support decides everything. On a guest that cannot use virtio-mem the hotplug does not fail — it silently stays pending:

Guest OS	Kernel / driver	Behavior (verified)
Debian 13	6.12	**Works.** `guestCurrent` reached the new size ~30 seconds after the patch, with all hotplugged memory onlined automatically and no manual step needed (unlike CPU hotplug). Note the automatic onlining is a property of the image, not of the virtio-mem driver: this image's kernel has automatic block onlining enabled (`/sys/devices/system/memory/auto_online_blocks` reports `online`). On a guest without such a policy (kernel default, boot parameter, or udev rule), hotplugged blocks stay offline and virtio-mem pauses further plugging — check `auto_online_blocks` if hotplug stalls on a different image.
CentOS 7.9	3.10	**Does not take effect.** No virtio-mem driver exists for this kernel. `guestRequested` stays above `guestCurrent` indefinitely; no error and no `RestartRequired` condition is raised.

Guest OS	Kernel / driver	Behavior (verified)
Ubuntu 20.04	5.4 (GA)	Same as CentOS 7 — the kernel predates virtio-mem entirely.
Ubuntu 20.04	5.15 (HWE)	**Still does not work**, in a more subtle way: the `virtio_mem` module loads, but the device rejects it — `dmesg` shows `virtio_mem virtio1: virtio: device refuses features: 3`, because the platform's QEMU requires the `UNPLUGGED_INACCESSIBLE` feature that this kernel lacks. Note Ubuntu 22.04's GA kernel is also 5.15 and does not carry the feature either — use the 22.04 HWE kernel or Ubuntu 24.04+.
Windows Server 2022	virtio-win `viomem`	**Works once the `viomem` driver is installed** — verified live: 8Gi → 10Gi applied online, with the guest reporting the new total immediately and no reboot. Without the driver the virtio-mem PCI device (`PCI\VEN_1AF4&DEV_1058`) sits in Device Manager with an error, and the request stays pending forever. The driver ships on the virtio-win ISO that the platform provides (verified with virtio-win 0.1.266): install it from an elevated shell, then retry or wait for the pending request to apply.

Installing the Windows driver from the attached virtio-win CD-ROM (drive letter may differ):

```
pnputil /add-driver E:\viomem\2k22\amd64\viomem.inf /install
# Get-PnpDevice | ? InstanceId -match 1058   ->  Status OK, "VirtIO Viome
m Driver"
```

The full virtio-win guest-tools installer (`virtio-win-guest-tools.exe` , on the same ISO) also installs `viomem` by default in current versions, so VMs provisioned with a recent guest-tools run get memory hotplug support out of the box. The driver exists for x64 (and ARM64 on newer Windows) only.

Notes on the pending state:

- The pending request is **not lost**. It is applied as ordinary boot memory on the next restart of the VM (`guestAtBoot` becomes the requested size — verified on CentOS 7). For guests without a working driver, a memory "hotplug" degrades gracefully into a planned

cold resize. Alternatively, a pending request can be **cancelled** by setting `memory.guest` back to the current live value — the revert is accepted without `RestartRequired` because nothing actually shrinks below boot.

- Administrators can confirm the mechanism from the node side: `virsh dumpxml <domain>` inside the virt-launcher pod shows a `<memory model='virtio-mem'>` device whose `requested` size follows each hotplug while `current` reflects what the guest driver actually accepted (`current: 0` on guests without working virtio-mem).

Scaling Down: What Works Live and What Needs a Restart

The two resources behave differently (verified on Linux and Windows guests):

- **Memory can be scaled down online — as a best-effort request, and only as far as the boot-time value.** Lowering `memory.guest` to any value ≥ `guestAtBoot` is *accepted* live: no `RestartRequired`, one automatic live migration, and the virtio-mem device is asked to give the memory back. Whether the guest fully complies is **not guaranteed** — it depends on how much of the hotplugged range the guest can actually free (memory pressure, unmovable kernel allocations, locked pages, and the onlining policy all play in). Always confirm the outcome by comparing `guestCurrent` with `guestRequested`. On idle guests the full shrink completed within a minute in our tests (4Gi → 3Gi on Debian 13, 10Gi → 9Gi → 8Gi on Windows Server 2022); on a Debian 13 guest under artificial memory pressure the same request stalled partway (`guestRequested: 2Gi`, `guestCurrent` stuck at ~2.1Gi) with no error and no condition — and a small remainder stayed unreclaimable even after the pressure was removed, because kernel allocations had landed in the hotplugged blocks. A partially completed shrink is not a failure state: the freed part is returned, the request stays pending, and you can revert the spec to the current live value at any time.
- **Lowering memory below `guestAtBoot`** is not possible live: the VM gets `RestartRequired` (`memory updated in template spec to a value lower than what the VM started with`) and nothing changes until a restart. In other words, live shrink can only try to reclaim previously hotplugged memory, never boot memory.
- **CPU cannot be scaled down live at all.** Any reduction of `sockets` sets `RestartRequired` (`Reduction of CPU socket count requires a restart`); the topology is untouched until a restart.

Reverting the spec back to the current live value clears a pending `RestartRequired` condition.

Verified Behavior Summary

Operation	Platform behavior	CentOS 7.9 (3.10)	Ubuntu 20.04 (5.4 / HWE 5.15)	Debian 13 (6.12)
CPU hotplug (sockets ↑)	Applied via automatic live migration	vCPU onlined automatically	Added but offline — manual online or udev rule	Added but offline — manual online or udev rule
Memory hotplug (guest ↑)	virtio-mem resize via automatic live migration	Pending forever; applies on next restart	Pending forever (5.15 fails feature negotiation); applies on next restart	Online in ~30s, blocks auto-onlined
Memory scale-down (guest ↓, ≥ boot value)	Requested live via automatic live migration; completion depends on the guest freeing the memory	— (no working driver)	— (no working driver)	Completed online when idle; stalls partially (no error) under memory pressure
Memory scale-down (guest ↓, < boot value)	Not supported live — `RestartRequired`	—	—	—
CPU scale-down (sockets ↓)	Not supported live — `RestartRequired`	—	—	—

Practical guidance for memory hotplug: what matters is whether the guest driver supports the `UNPLUGGED_INACCESSIBLE` virtio-mem feature, and distribution kernels backport it

independently of their version number — so do not judge by `uname -r` alone. Known-good guests include Debian 12+, Ubuntu 24.04+ (or 22.04 with the HWE kernel — the GA 5.15 kernel does not work), and RHEL-family releases that backport the feature (Red Hat documents virtio-mem guest support for RHEL 9.4+ and 8.10 — but note RHEL 8.10 supports hot-*plug* only by default: memory unplugging is disabled unless the guest boots with `memhp_default_state=online_movable`); the tested Windows Server 2022 guest works once the `viomem` driver from the platform's virtio-win ISO is installed. When in doubt, test one hotplug and check `dmesg` / Device Manager as shown above. For CPU hotplug, the hypervisor-side hot-add succeeded on all four guests tested here; whether the guest uses the vCPUs without extra steps varies — RHEL-family and Windows Server 2022 did, Debian/Ubuntu need the udev rule above, and other Windows editions may differ (Microsoft ties CPU hot-add to dynamic-hardware-partitioning support in specific Server SKUs — verify on your edition before relying on it).

Troubleshooting

- **Hotplug appears stuck** — check that the VM can live-migrate (`LiveMigratable` condition on the VMI, migration events in the namespace). Hotplug is delivered by live migration, so anything that blocks migration (RWO-backed PVC disks, node constraints) blocks hotplug. The condition message shows only the first blocking reason — fixing one may reveal another.
- **`guestRequested` > `guestCurrent` (a grow request is stuck)** — the guest is not taking the memory. On Linux, run `dmesg | grep virtio_mem` : `device refuses features` means the driver lacks the `UNPLUGGED_INACCESSIBLE` feature; no output at all means the driver never bound — a kernel without virtio-mem support, but also possibly a module that simply is not loaded, so check `lsmod | grep virtio_mem` before concluding the kernel is too old. If the driver is fine, check the onlining policy (`cat /sys/devices/system/memory/auto_online_blocks` ; virtio-mem pauses plugging while added blocks stay offline). On Windows, a stuck grow means the `viomem` driver is missing, too old, not bound, or in an error state — Device Manager shows the virtio-mem PCI device (`PCI\VEN_1AF4&DEV_1058`) as errored in any of these cases. Installing the current driver from the platform's virtio-win ISO as shown above covers the common cases; the pending request then applies without a reboot. Either way the request is not lost — upgrade the guest, or plan a restart to apply the memory cold.

- `guestRequested` < `guestCurrent` **(a shrink request is stuck)** — usually the guest cannot free that part of the hotplugged range (memory in active use, locked pages, or unmovable kernel allocations that landed in the hotplugged blocks) — but first confirm the guest actually supports *and has enabled* memory unplugging: some guests that hot-plug fine ship with unplugging disabled by default (RHEL 8.10, for example, requires booting with `memhp_default_state=online_movable`). The request keeps retrying in the background and may complete later, complete partially, or never fully complete. Reduce memory pressure in the guest, accept the partial result, or revert `memory.guest` to the current live value to cancel the remainder.
- **New vCPU not visible in** `nproc` — it is hot-added but offline. Check `cat /sys/devices/system/cpu/online` vs `offline`, and online it manually or via udev rule as shown above.
- `RestartRequired` **on the VM** — the requested change cannot be applied live (CPU reduction, memory below the boot value, first-time addition of a `cpu` block, or raising `maxSockets` / `maxGuest`). Either revert the spec to clear it, or restart the VM to apply it.

Virtual Machine Recovery

In certain scenarios, such as incorrect modifications to fstab or filesystem errors requiring fsck, virtual machines may fail to start properly. In such cases, you can utilize rescue mode to repair the root filesystem (rootfs) or retrieve data from the system.

TOC

Steps to Operate

Obtain Image Address

1. In the left navigation bar, click **Virtualization Management** > **Virtual Machine Images**.
2. Select the platform-provided **Source** as **Image Repository**, and the **Operating System** as either **CentOS or Ubuntu**. Click ⋮ > **Update** on the right.
3. Copy and save the **Image Address**. This document uses `192.168.1.1:11443/3rdparty/vmdisks/centos:7.9` as an example.

4. Click **Cancel**.

Modify Virtual Machine YAML File

1. Access the **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click ⋮ > **Stop** on the right side of the virtual machine that needs repair to **Stop** or **Force Stop** it.
4. Click ⋮ > **Update** on the right side of the virtual machine.
5. Switch to **YAML** and modify the following fields.

   - Add the following content under `spec.template.spec.domain.devices.disks`. Adding a bootOrder parameter can control which disk is prioritized during the virtual machine's boot process; a lower bootOrder value indicates higher priority.
     Note: If the original `spec.template.spec.domain.devices.disks` field contains `bootOrder: 1`, increase the original value to ensure that the newly added bootOrder value is lower than the original.

     ```
     disks:
       - bootOrder: 1
         disk:
           bus: virtio
         name: containerdisk
     ```

     Modified YAML example:

```
domain:
  devices:
    disks:
      - bootOrder: 1  # Added Field
        disk:
          bus: virtio
        name: containerdisk
      - disk:
          bus: virtio
        name: cloudinitdisk
      - disk:      # Increase the original bootOrder: 1 value
          bus: virtio
        name: rootfs
        bootOrder: 10
      - disk:
          bus: virtio
        name: "1"
```

- Add the following content under `spec.template.spec.volumes`.

  Note: Please replace the image address in the following `image` field with the one obtained from Obtain Image Address.

  ```
  - containerDisk:
      image: 192.168.1.1:11443/3rdparty/vmdisks/centos:7.9
    name: containerdisk
  ```

  Modified YAML example:

  ```
  volumes:
    - containerDisk:  # Added Field
        image: 192.168.1.1:11443/3rdparty/vmdisks/centos:7.9
      name: containerdisk
    - dataVolume:
        name: k2-rootfs
      name: rootfs
    - dataVolume:
        name: k2-1
      name: "1"
  ```

6. Click **Update**.

Note: After modifying the YAML file, do not switch to **Form**, just click **Update** directly.

7. Click ⋮ > **Start** on the right side of the virtual machine.

Mount the Original rootfs and Perform Repair

1. Log in to the virtual machine using the original password or key and enter the command `df -h /` to find that the rootfs filesystem has been replaced. You can use mount-related commands to mount it or fsck-related commands to check and repair the original filesystem.
2. After completion, shut down the virtual machine.

Restore the Virtual Machine YAML File

Follow the steps in Modify Virtual Machine YAML File to restore the virtual machine YAML file to its original state. At this point, the virtual machine can start normally.

Clone Virtual Machines on KubeVirt

This document provides step-by-step guidance on cloning virtual machines (VMs) using KubeVirt's `VirtualMachineClone` API.

TOC

Ensure Prerequisites

Before initiating a VM clone operation, make sure the following requirements are satisfied:

- **Snapshot-Capable Storage**: The Clone API relies on Snapshot & Restore functionalities. The virtual machine's storage class must support **volume snapshots**, and snapshot functionality must be explicitly enabled for that storage backend.

Start Quickly

Follow these quick steps to clone a VM:

1. **Prepare the Clone Manifest**:

   Create a file named `clone.yaml` with the following structure:

   ```yaml
   apiVersion: clone.kubevirt.io/v1beta1
   kind: VirtualMachineClone
   metadata:
     name: example-vm-clone
     namespace: ns-where-vm-run
   spec:
     source:
       apiGroup: kubevirt.io
       kind: VirtualMachine
       name: source-vm
     target:
       apiGroup: kubevirt.io
       kind: VirtualMachine
       name: target-vm
   ```

2. **Execute the Clone Operation**:

   Apply the manifest:

   ```bash
   kubectl create -f clone.yaml
   ```

3. **Monitor the Clone Status**:

   Wait until the cloning is completed successfully:

   ```bash
   kubectl wait vmclone example-vm-clone --for=jsonpath='{.status.phase}'=
   Succeeded --timeout=300s
   ```

4. **Verify the Cloned VM**:

   Inspect the cloned VM configuration:

   ```bash
   kubectl get vm target-vm -o yaml
   ```

5. **Fix the DataVolume Label (UI metadata)**:
   The platform UI links VMs to their disks through the label `vm.cpaas.io/used-by=<vm-name>` that is automatically added to every DataVolume. After a clone operation the new DataVolume inherits the label from the *source* VM, so the UI still thinks it belongs to the old VM. Update the label on the newly created DV so the relationship displays correctly (functionality is **not** affected).

   ```
   # List DataVolumes in the VM's namespace; the cloned DV name usually st
   arts with "restore-"
   kubectl get datavolumes -n <ns-where-vm-run>

   # Overwrite the label to point to the cloned VM
   kubectl label datavolume <new-dv-name> -n <ns-where-vm-run> vm.cpaas.i
   o/used-by=<target-vm> --overwrite
   ```

Understand the VirtualMachineClone Object

View a Complete VirtualMachineClone Example

Here's a complete example of a `VirtualMachineClone` resource with detailed inline comments:

```yaml
apiVersion: clone.kubevirt.io/v1beta1
kind: VirtualMachineClone
metadata:
  name: detailed-vm-clone
  namespace: ns-where-vm-run
spec:
  # Source VM details
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: vm-source

  # Target VM details
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: vm-target

  # Filters for labels and annotations copied from source
  labelFilters:
    - "*"
    - "!exclude-key/*"
  annotationFilters:
    - "include-annotations/*"

  # Template filters to manage network annotations.
  # annotationFilters is a WHITELIST: list "*" first to keep all annotati
ons,
  # then negate the ones to drop. On Kube-OVN, drop the per-VM OVN annota
tions
  # (fixed IP / pod-bridge live-migration) so the clone gets a fresh iden
tity:
  template:
    labelFilters:
      - "*"
    annotationFilters:
      - "*"
      - "!ovn.kubernetes.io/*"

  # Explicitly set new MAC addresses
  newMacAddresses:
    eth0: "02-00-00-aa-bb-cc"
```

```
  # Explicitly set SMBios serial
  newSMBiosSerial: "unique-serial-1234"

  # JSON patches to further customize the cloned VM
  patches:
    - '{"op": "add", "path": "/metadata/labels/new-label", "value": "new-value"}'
    - '{"op": "replace", "path": "/spec/template/metadata/annotations/new-annotation", "value": "updated-value"}'
```

Understand Each Field

- **Source and Target**:
  - Define the original VM (`source`) and the cloned VM (`target`).
  - Auto-generated if the `target` name is omitted.
  - Both VMs must reside within the same namespace.
- **Label and Annotation Filters**:
  - Control copying or excluding labels/annotations from the source VM using wildcards (`*`) and negations (`!`).
- **Template Label and Annotation Filters**:
  - Useful for managing network-related annotations, especially with CNIs like Kube-OVN.
- **newMacAddresses**:
  - Optionally specify new MAC addresses for network interfaces.
  - Automatically regenerated if omitted.
- **newSMBiosSerial**:
  - Optionally specify a new SMBios serial.
  - Auto-generated based on VM name if not provided.
- **JSON Patches**:
  - Advanced customizations directly applied to VM specs.

Check Clone Operation Phases

The `.status.phase` of a `VirtualMachineClone` object changes according to the cloning process progress. The table below explains each phase:

Phase	Explanation
SnapshotInProgress	Creating a snapshot of the source VM, initial step when cloning a running VM.
CreatingTargetVM	Snapshot is complete; creating metadata and specification for the target VM.
RestoreInProgress	DataVolume and PersistentVolumeClaim creation in progress, restoring data from snapshot.
Succeeded	Operation successfully completed. Target VM and storage are ready.
Failed	Operation failed. Check `events` and `status.conditions` for detailed error information.
Unknown	Unable to determine the clone operation status, potentially indicating a controller issue.

Migrate Virtual Machines Between Storage Classes

This document describes how to migrate a virtual machine (VM) disk from one StorageClass to another. Use this procedure when you need to move VM storage to another backend, storage pool, or CSI driver, for example from Ceph RBD to CephFS, between two Ceph RBD pools, or from platform-provided storage to a customer-provided storage class.

Two migration methods are available:

Method	VM status	Recommended scenario
Cold migration with a CDI clone	Stopped	Use when the source or destination storage supports only `ReadWriteOnce`, when local or topology-aware storage is involved, or when maximum compatibility is more important than avoiding downtime.
Live storage migration	Running	Use only when the VM, source storage, destination storage, network, and target node scheduling conditions are all live-migratable.

TOC

Prerequisites

- ACP virtualization is enabled and includes live storage migration support.
- You have permission to manage `VirtualMachine`, `VirtualMachineInstance`, `PersistentVolumeClaim`, and `DataVolume` resources in the VM namespace.
- The source VM uses a CDI `DataVolume` or a PVC-backed disk.
- The destination StorageClass has enough capacity and supports the required `accessModes` and `volumeMode`.
- For cold migration, plan a maintenance window because the VM is stopped while the disk is cloned and the VM definition is updated.
- For live storage migration, the running VMI must report both `LiveMigratable=True` and `StorageLiveMigratable=True`.

Prepare the Migration

Set variables for the VM and destination storage:

```
NAMESPACE=<vm-namespace>
VM=<vm-name>
ROOT_VOLUME=<root-disk-volume-name>
SRC_PVC=<source-pvc-name>
DST_SC=<destination-storage-class>
```

Check the destination StorageProfile before creating the destination disk:

```
kubectl get storageprofile "$DST_SC" -o yaml
```

Choose `accessModes` and `volumeMode` from the claim property sets supported by the destination StorageProfile. For example, Ceph RBD is commonly used with `volumeMode: Block`, while CephFS is commonly used with `volumeMode: Filesystem`.

For a `VirtualMachineStorageMigrationPlan`, which is used for live storage migration and described later in this document, you can also set `accessModes` or `volumeMode` to `Auto` and let the storage migration controller look up the value from the StorageProfile. For a manually created `DataVolume`, use explicit Kubernetes PVC values such as `ReadWriteOnce`, `ReadWriteMany`, `Block`, or `Filesystem`.

If you plan to use live storage migration, check the running VMI conditions:

```
kubectl get vmi "$VM" -n "$NAMESPACE" \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status}{" "}{.reason}{"\n"}{end}'
```

Continue with live storage migration only when the output includes both of the following conditions:

```
LiveMigratable=True
StorageLiveMigratable=True
```

If either condition is missing or set to `False`, use cold migration or resolve the reported limitation before retrying live migration.

Cold Migration with a DataVolume Clone

Cold migration is the most compatible method across CSI drivers. It stops the VM, clones the source PVC into a new DataVolume that uses the destination StorageClass, and then updates the VM to use the new DataVolume.

Stop the VM

Stop the VM:

```
kubectl patch vm "$VM" -n "$NAMESPACE" --type=merge \
  -p '{"spec":{"running":false}}'
```

> **Note**
>
> Some VM API versions warn that `spec.running` is deprecated in favor of `spec.runStrategy`. If the VM already uses `spec.running`, do not set `spec.runStrategy` without first removing `spec.running`, because the two fields are mutually exclusive.

Wait until the VMI is deleted:

```
kubectl wait vmi "$VM" -n "$NAMESPACE" --for=delete --timeout=10m
```

Create the Destination DataVolume

Create a DataVolume that clones the source PVC into the destination StorageClass:

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <destination-datavolume-name>
  namespace: <vm-namespace>
spec:
  source:
    pvc:
      namespace: <vm-namespace>
      name: <source-pvc-name>
  pvc:
    accessModes:
      - <destination-access-mode>
    volumeMode: <Block-or-Filesystem>
    resources:
      requests:
        storage: <destination-size>
    storageClassName: <destination-storage-class>
```

Apply the manifest:

```bash
kubectl apply -f <destination-datavolume-file>.yaml
```

For a StorageClass with `volumeBindingMode: Immediate`, wait for the clone to complete:

```bash
kubectl wait dv <destination-datavolume-name> -n "$NAMESPACE" \
  --for=jsonpath='{.status.phase}'=Succeeded --timeout=60m
```

For a StorageClass with `volumeBindingMode: WaitForFirstConsumer`, the DataVolume can remain in `PendingPopulation` until the VM starts and becomes the first consumer. This is expected for local or topology-aware storage. Continue with the VM update, start the VM, and monitor the DataVolume and VMI.

Note

When migrating from block mode to filesystem mode, request a destination size larger than the source disk. A 5% to 10% increase is a practical starting point because the destination filesystem and disk image file need additional space.

Update the VM Disk Reference

Update the VM so the root disk volume points to the destination DataVolume.

When editing arrays such as `spec.dataVolumeTemplates` and `spec.template.spec.volumes`, keep all required entries. If the VM has other disks, cloud-init volumes, secrets, or config maps, include them in the final VM specification so they are not removed.

Example patch content:

```
spec:
  dataVolumeTemplates:
    - metadata:
        name: <destination-datavolume-name>
      spec:
        source:
          pvc:
            namespace: <vm-namespace>
            name: <source-pvc-name>
        pvc:
          accessModes:
            - <destination-access-mode>
          volumeMode: <Block-or-Filesystem>
          resources:
            requests:
              storage: <destination-size>
          storageClassName: <destination-storage-class>
  template:
    spec:
      volumes:
        - name: <root-disk-volume-name>
          dataVolume:
            name: <destination-datavolume-name>
        - name: <other-volume-name>
          <other-volume-source>: {}
```

Apply the patch:

```
kubectl patch vm "$VM" -n "$NAMESPACE" --type=merge --patch-file=<vm-patch-file>.yaml
```

Start and Verify the VM

Start the VM:

```
kubectl patch vm "$VM" -n "$NAMESPACE" --type=merge \
  -p '{"spec":{"running":true}}'
```

Check the VM, VMI, PVC, and DataVolume status:

```
kubectl get vm,vmi,pvc,dv -n "$NAMESPACE"
```

If the destination uses `WaitForFirstConsumer`, monitor the DataVolume and PVC until the VM starts:

```
kubectl get dv <destination-datavolume-name> -n "$NAMESPACE" -w
kubectl describe pvc <destination-pvc-name> -n "$NAMESPACE"
```

After the VM is running and guest workload checks pass, delete the old DataVolume or PVC only after confirming that it is no longer referenced by the VM.

```
kubectl delete dv <old-datavolume-name> -n "$NAMESPACE"
```

Live Storage Migration

Use live storage migration only when the VM must stay running and all live migration prerequisites are met.

Confirm Live Migration Readiness

Check the VMI conditions:

```
kubectl get vmi "$VM" -n "$NAMESPACE" \
  -o jsonpath='{range .status.conditions[*]}{.type}={.status}{" "}{.reason}{"\n"}{end}'
```

Proceed only when the output includes `LiveMigratable=True` and `StorageLiveMigratable=True`.

Also confirm the following requirements:

- The destination StorageClass can create a PVC with the selected `accessModes` and `volumeMode`.
- The VM can be scheduled on more than one node.
- The VM does not report `RestartRequired=True` in `.status.conditions`. If the VM has pending changes that require a restart, restart the VM or resolve those pending changes before starting live storage migration.
- If the cluster has mixed CPU models, the VM uses a CPU model supported by all candidate nodes.
- If the VM uses bridge binding on the pod network, the VM has the following annotation:

```
kubevirt.io/allow-pod-bridge-network-live-migration: 'true'
```

Create the Migration Objects

The ACP console is the preferred entry point because it validates the VM and destination storage options before creating the migration resources.

If you use Kubernetes manifests, create a `VirtualMachineStorageMigrationPlan` and a `VirtualMachineStorageMigration`. The plan alone does not execute — the migration object is the trigger:

```yaml
apiVersion: migrations.kubevirt.io/v1alpha1
kind: VirtualMachineStorageMigrationPlan
metadata:
  name: <migration-plan-name>
  namespace: <vm-namespace>
spec:
  # keepSource (default) keeps the original PVCs after a successful migra
tion;
  # the source storage is NOT auto-deleted — remove it manually once veri
fied.
  retentionPolicy: keepSource
  virtualMachines:
    - name: <vm-name>
      targetMigrationPVCs:
        - volumeName: <root-disk-volume-name>
          destinationPVC:
            name: <destination-pvc-name>
            storageClassName: <destination-storage-class>
            accessModes:
              - <destination-access-mode>
            volumeMode: <Block-Filesystem-or-Auto>
---
apiVersion: migrations.kubevirt.io/v1alpha1
kind: VirtualMachineStorageMigration
metadata:
  name: <migration-name>
  namespace: <vm-namespace>
spec:
  virtualMachineStorageMigrationPlanRef:
    apiVersion: migrations.kubevirt.io/v1alpha1
    kind: VirtualMachineStorageMigrationPlan
    name: <migration-plan-name>
    namespace: <vm-namespace>
```

Apply the manifest:

```bash
kubectl create -f <storage-migration-file>.yaml
```

Monitor progress:

```
kubectl get virtualmachinestoragemigration,virtualmachinestoragemigration
plan \
  -n "$NAMESPACE" -w

kubectl get vmim,pvc,dv -n "$NAMESPACE"
```

The expected flow is:

1. ACP creates the destination PVC and DataVolume.
2. ACP updates the VM disk reference to the destination DataVolume.
3. KubeVirt performs a workload update through live migration.
4. ACP cleans up the completed workload update resources.
5. The `VirtualMachineStorageMigration` reaches `Completed`.

When the migration reaches `Completed`, the VM specification references the destination DataVolume and the VM continues running on the migrated disk. Verify the VM, VMI, destination PVC, and destination DataVolume before deleting any old storage objects:

```
kubectl get vm "$VM" -n "$NAMESPACE" \
  -o jsonpath='{range .spec.template.spec.volumes[*]}{.name}{" -> "}{.dat
aVolume.name}{"\n"}{end}'

kubectl get vm,vmi,pvc,dv -n "$NAMESPACE"
```

After the migration succeeds and guest workload checks pass, remove the old DataVolume or PVC only if it is no longer referenced by the VM or by any rollback procedure.

Troubleshooting

No VM Is Ready for Live Storage Migration

Describe the migration plan:

```
kubectl describe virtualmachinestoragemigrationplan <migration-plan-name>
-n "$NAMESPACE"
```

If the plan reports `No virtual machines are ready for storage migration`, check whether the VM is running and whether the VMI reports both `LiveMigratable=True` and `StorageLiveMigratable=True`.

If either condition is false, use cold migration or fix the underlying reason first. Also confirm that the target `accessModes` and `volumeMode` are supported by the destination StorageProfile, and that the VM does not report `RestartRequired=True`. Common causes include non-shared PVCs, local storage, unsupported network binding, unsupported target PVC properties, pending VM changes that require a restart, and node scheduling constraints.

Live Migration Is Stuck in Scheduling

Describe the pending launcher pod or the `VirtualMachineInstanceMigration`:

```
kubectl get pod -n "$NAMESPACE" | grep virt-launcher
kubectl describe vmim <vmim-name> -n "$NAMESPACE"
```

Common causes include:

- The destination launcher pod cannot be scheduled on a node different from the source node.
- The VM uses a CPU model or CPU feature set that exists only on the source node.
- The destination PVC cannot be attached or mounted on the target node.

Use a portable CPU model and confirm that the destination storage supports the required access mode.

Destination DataVolume Does Not Complete

Inspect the DataVolume and PVC events:

```
kubectl describe dv <destination-datavolume-name> -n "$NAMESPACE"
kubectl describe pvc <destination-pvc-name> -n "$NAMESPACE"
```

For `WaitForFirstConsumer` storage, `PendingPopulation` can be expected until a consumer pod is scheduled. For block-to-filesystem migration, increase the requested destination size and retry if CDI reports that the target PVC is too small.

Live Storage Migration Does Not Reach Completed

If the `VirtualMachineStorageMigration` does not reach `Completed`, inspect both the storage migration resource and the underlying KubeVirt migration:

```
kubectl describe virtualmachinestoragemigration <migration-name> -n "$NAM
ESPACE"
kubectl get vmim -n "$NAMESPACE"
kubectl describe vmim <vmim-name> -n "$NAMESPACE"
```

If the `VirtualMachineInstanceMigration` is still running, continue troubleshooting it as a live migration issue. If the `VirtualMachineInstanceMigration` has already succeeded but the storage migration resource still does not complete, collect the migration resource, migration plan, VM, VMI, PVC, DataVolume, and controller logs before contacting support.

Migration Resources Are Stuck During Cleanup

Delete the migration resources first:

```
kubectl delete virtualmachinestoragemigration <migration-name> -n "$NAMES
PACE" --wait=false
kubectl delete virtualmachinestoragemigrationplan <migration-plan-name> -
n "$NAMESPACE" --wait=false
```

If a finalizer prevents deletion, remove it only after confirming that no active migration is still running:

```
kubectl patch virtualmachinestoragemigration <migration-name> -n "$NAMESP
ACE" --type=json \
  -p='[{"op":"remove","path":"/metadata/finalizers"}]'

kubectl patch virtualmachinestoragemigrationplan <migration-plan-name> -n
"$NAMESPACE" --type=json \
  -p='[{"op":"remove","path":"/metadata/finalizers"}]'
```

Before restarting the VM after an interrupted live storage migration, confirm which DataVolume the VM references:

```
kubectl get vm "$VM" -n "$NAMESPACE" \
  -o jsonpath='{range .spec.template.spec.volumes[*]}{.name}{" -> "}{.dat
aVolume.name}{"\n"}{end}'
```

If the VM specification was updated but the migration did not complete, either finish the migration or restore the VM disk reference to the intended DataVolume before starting the VM.

Configuring High Availability for Virtual Machines

TOC

Run Strategy and Eviction Strategy

Two virtual machine spec fields determine how a virtual machine behaves on failure and during node maintenance:

- **Run strategy** (`spec.runStrategy`) controls the desired power state and restart behavior: `Always` (keep it running, restart if it stops), `RerunOnFailure` (restart only after a failure), `Manual` (state is controlled only by start/stop/restart), and `Halted` (keep it stopped). ACP virtual machines use the simpler `spec.running` boolean (`true` / `false`), which is equivalent to `Always` / `Halted` .
- **Eviction strategy** (`spec.template.spec.evictionStrategy`) controls what happens when a node is drained for maintenance or an upgrade:
  - `LiveMigrate` — the virtual machine is live-migrated to another node instead of being killed.
  - `LiveMigrateIfPossible` — live-migrate if the virtual machine is migratable, otherwise behave as `None` .
  - `None` — take no special action (the virtual machine is shut down with the node).

  A **non-migratable** virtual machine (for example one using RWO storage or a passed-through device) with `evictionStrategy: LiveMigrate` will **block a node drain** — and therefore a cluster upgrade — until it is stopped or migrated manually. Plan RWX storage (see Migrate Virtual Machines Between Storage Classes) for virtual machines that must survive maintenance, or set `LiveMigrateIfPossible` to avoid blocking.

Overview

Hardware is imperfect and software contains bugs. When node-level failures, such as the kernel hangs or network interface controllers (NICs) fail, the work required from the cluster does not decrease, and workloads from affected nodes need to be restarted somewhere. However, some workloads, such as ReadWriteOnce (RWO) volumes and StatefulSets, might require at-most-one semantics.

Failures affecting these workloads risk data loss, corruption, or both. It is important to ensure that the node reaches a safe state, known as fencing before initiating recovery of the workload, known as remediation and ideally, recovery of the node also.

It is not always practical to depend on administrator intervention to confirm the true status of the nodes and workloads. To facilitate such intervention, Alauda Container Platform provides multiple components for the automation of failure detection, fencing and remediation.

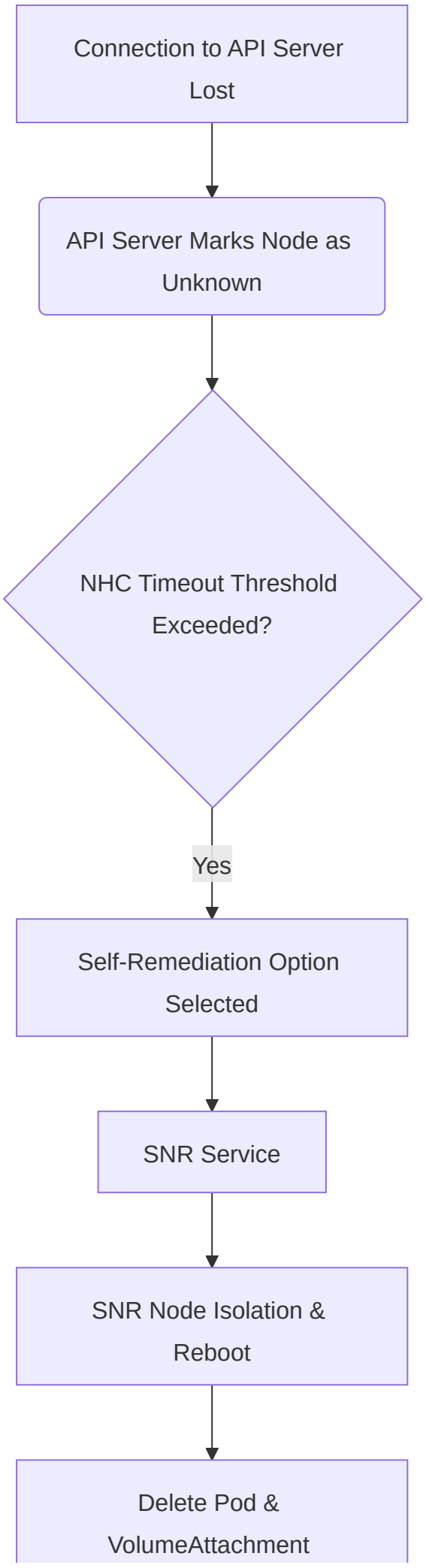
Glossary

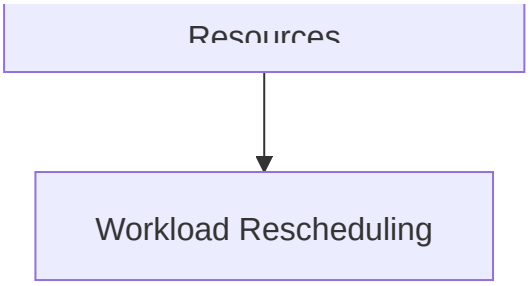
Acronym	Term
SNR	Self Node Remediation
NHC	Node Health Check

Component Overview

- **Self Node Remediation Operator**
  The Self Node Remediation Operator is a Alauda Container Platform add-on Operator that implements an external system of fencing and remediation that reboots unhealthy nodes and deletes resources, such as Pods and VolumeAttachments. The reboot ensures that the workloads are fenced, and the resource deletion accelerates the rescheduling of affected workloads. Unlike other external systems, Self Node Remediation does not require any management interface, like, for example, Intelligent Platform Management Interface (IPMI) or an API for node provisioning.
  Self Node Remediation can be used by failure detection systems, like Machine Health Check or Node Health Check.
- **Node Health Check Operator**
  The Node Health Check Operator is a Alauda Container Platform add-on Operator that implements a failure detection system that monitors node conditions. It does not have a built-in fencing or remediation system and so must be configured with an external system that provides these features. By default, it is configured to utilize the Self Node Remediation system.

Flow of events during fencing and remediation


Connection to API Server Lost
API Server Marks Node as Unknown
NHC Timeout Threshold Exceeded?
Yes
Self-Remediation Option Selected
SNR Service
SNR Node Isolation & Reboot
Delete Pod & VolumeAttachment



Procedure

1 Operator Listing

- **Download** the **Alauda Build of SelfNodeRemediation** installation package corresponding to your platform architecture.
- **Upload** the **Alauda Build of SelfNodeRemediation** installation package using the Upload Packages mechanism.
- **Download** the **Alauda Build of NodeHealthCheck** installation package corresponding to your platform architecture.
- **Upload** the **Alauda Build of NodeHealthCheck** installation package using the Upload Packages mechanism.

2 Deploying Self Node Remediation Operator

1. Login, go to the **Administrator** page.
2. Click **Marketplace** > **OperatorHub** to enter the **OperatorHub** page.
3. Find the **Alauda Build of SelfNodeRemediation**, click **Install**, and navigate to the **Install Alauda Build of SelfNodeRemediation** page.
   Configuration Parameters:

Parameter	Recommended Configuration
Channel	The default channel is `stable`.
Installation Mode	`Cluster`: All namespaces in the cluster share a single Operator instance for creation and management, resulting in lower resource usage.

Parameter	Recommended Configuration
Installation Place	Select `Recommended`, Namespace only support **workload-availability**.
Upgrade Strategy	`Manual`: When there is a new version in the Operator Hub, manual confirmation is required to upgrade the Operator to the latest version.

3 Configuring Self Node Remediation Operator(optional)

The Self Node Remediation Operator creates the `SelfNodeRemediationConfig` CR with the name `self-node-remediation-config`. The CR is created in the namespace of the Self Node Remediation Operator.

> **Note**
>
> A change in the `SelfNodeRemediationConfig` CR re-creates the Self Node Remediation daemon set.

The `SelfNodeRemediationConfig` CR resembles the following YAML file:

```yaml
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationConfig
metadata:
  name: self-node-remediation-config
  namespace: workload-availability
spec:
  safeTimeToAssumeNodeRebootedSeconds: 180
  watchdogFilePath: /dev/watchdog
  isSoftwareRebootEnabled: true
  apiServerTimeout: 15s
  apiCheckInterval: 5s
  maxApiErrorThreshold: 3
  peerApiServerTimeout: 5s
  peerDialTimeout: 5s
  peerRequestTimeout: 5s
  peerUpdateInterval: 15m
  hostPort: 30001
  customDsTolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io.infra
    operator: Equal
    value: "value1"
    tolerationSeconds: 3600
```

Parameters

Parameter	Description
safeTimeToAssumeNodeRebootedSeconds	Specify an optional time duration that the Operator waits before recovering affected workloads running on an unhealthy node. Starting replacement pods while they are still running on the failed node can lead to data corruption and a violation of run-once semantics. The Operator calculates a minimum duration using the values in the ApiServerTimeout, ApiCheckInterval, MaxApiErrorThreshold, PeerDialTimeout, and PeerRequestTimeout fields, as well as the watchdog timeout and the cluster size at the time of remediation.

Parameter	Description
watchdogFilePath	Specify the file path of the watchdog device in the nodes. If you enter an incorrect path to the watchdog device, the Self Node Remediation Operator automatically detects the softdog device path. If a watchdog device is unavailable, the `SelfNodeRemediationConfig` CR uses a software reboot.
isSoftwareRebootEnabled	Specify if you want to enable software reboot of the unhealthy nodes. By default, the value of `isSoftwareRebootEnabled` is set to `true`. To disable the software reboot, set the parameter value to `false`.
apiServerTimeout	Specify the timeout duration to check connectivity with each API server. When this duration elapses, the Operator starts remediation. The timeout duration must be greater than or equal to 10 milliseconds.
apiCheckInterval	Specify the frequency to check connectivity with each API server. The timeout duration must be greater than or equal to 1 second.
maxApiErrorThreshold	Specify a threshold value. After reaching this threshold, the node starts contacting its peers. The threshold value must be greater than or equal to 1 second.
peerApiServerTimeout	Specify the duration of the timeout for the peer to connect the API server. The timeout duration must be greater than or equal to 10 milliseconds.
peerDialTimeout	Specify the duration of the timeout for establishing connection with the peer. The

Parameter	Description
	timeout duration must be greater than or equal to 10 milliseconds.
peerRequestTimeout	Specify the duration of the timeout to get a response from the peer. The timeout duration must be greater than or equal to 10 milliseconds.
peerUpdateInterval	Specify the frequency to update peer information such as IP address. The timeout duration must be greater than or equal to 10 seconds.
hostPort	Specify an optional value to change the port that Self Node Remediation agents use for internal communication. The value must be greater than 0. The default value is port 30001.
customDsTolerations	Specify custom toleration Self Node Remediation agents that are running on the DaemonSets to support remediation for different types of nodes.

Note

- The Self Node Remediation Operator creates the CR by default in the deployment namespace.
- The name for the CR must be `self-node-remediation-config`.
- You can only have one `SelfNodeRemediationConfig` CR.
- Deleting the `SelfNodeRemediationConfig` CR disables Self Node Remediation.

4 Configuring Self Node Remediation Template(optional)

The Self Node Remediation Operator also creates the SelfNodeRemediationTemplate Custom Resource Definition (CRD). This CRD defines the remediation strategy for the

nodes that is aimed to recover workloads faster. The following remediation strategies are available:

- **Automatic**
  This remediation strategy simplifies the remediation process by letting the Self Node Remediation Operator decide on the most suitable remediation strategy for the cluster. This strategy checks if the `OutOfServiceTaint` strategy is available on the cluster. If the `OutOfServiceTaint` strategy is available, the Operator selects the `OutOfServiceTaint` strategy. If the `OutOfServiceTaint` strategy is not available, the Operator selects the `ResourceDeletion` strategy. `Automatic` is the default remediation strategy.
- **ResourceDeletion**
  This remediation strategy removes the pods on the node, rather than the removal of the node object.
- **OutOfServiceTaint**
  This remediation strategy implicitly causes the removal of the pods and associated volume attachments on the node, rather than the removal of the node object. It achieves this by placing the `OutOfServiceTaint` strategy on the node.

The Self Node Remediation Operator creates the SelfNodeRemediationTemplate CR for the strategy self-node-remediation-automatic-strategy-template, which the Automatic remediation strategy uses.

The SelfNodeRemediationTemplate CR resembles the following YAML file:

```yaml
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  creationTimestamp: "2022-03-02T08:02:40Z"
  name: self-node-remediation-<remediation_object>-deletion-template
  namespace: workload-availability
spec:
  template:
    spec:
      remediationStrategy: <remediation_strategy>
```

Parameters

Parameter	Description
remediation_strategy	Values: Automatic、ResourceDeletion、OutOfServiceTaint

5 Deploying Node Health Check Operator

1. Login, go to the **Administrator** page.
2. Click **Marketplace** > **OperatorHub** to enter the **OperatorHub** page.
3. Find the **Alauda Build of NodeHealthCheck**, click **Install**, and navigate to the **Install Alauda Build of NodeHealthCheck** page.
   Configuration Parameters:

Parameter	Recommended Configuration
Channel	The default channel is `stable`.
Installation Mode	`Cluster`: All namespaces in the cluster share a single Operator instance for creation and management, resulting in lower resource usage.
Installation Place	Select `Recommended`, Namespace only support **workload-availability**.
Upgrade Strategy	`Manual`: When there is a new version in the Operator Hub, manual confirmation is required to upgrade the Operator to the latest version.

6 Create NodeHealthCheck instance

Execute the following command on the cluster control node:

Command

```
cat << EOF | kubectl apply -f -
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-<name>
spec:
  minHealthy: <minHealthy>
  remediationTemplate:
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    kind: SelfNodeRemediationTemplate
    name: self-node-remediation-automatic-strategy-template
    namespace: workload-availability
  selector: <selector>
  unhealthyConditions:
    - duration: 300s
      status: 'False'
      type: Ready
    - duration: 300s
      status: Unknown
      type: Ready
EOF
```

Example

```
cat << EOF | kubectl apply -f -
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-worker
spec:
  minHealthy: 51%
  remediationTemplate:
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    kind: SelfNodeRemediationTemplate
    name: self-node-remediation-automatic-strategy-template
    namespace: workload-availability
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  unhealthyConditions:
    - duration: 300s
      status: 'False'
      type: Ready
    - duration: 300s
      status: Unknown
      type: Ready
EOF
```

Parameters:

Parameter	Description
name	resource name
minHealthy	Specify the minimum proportion of healthy nodes. Faulty nodes will only be repaired when the proportion of healthy nodes is greater than or equal to this value. The default value is 51%
selector	Specify LabelSelector to match the nodes to be inspected and self-repaired. Please avoid specifying control-plane and worker nodes simultaneously in the same instance

7 Verification(optional)

Simulate the failure of the running node of the virtual machine and confirm that the virtual machine is automatically scheduled to run on other nodes.

Create a VM Template from an Existing Virtual Machine

This document outlines how to create a reusable virtual machine (VM) template from an existing VM for rapid deployment of new VMs.

TOC

Prerequisites

- A properly deployed and configured KubeVirt environment.
- Access to the Web Console and kubectl tool.
- A configured VM with necessary software already installed.

Procedure

Step 1: Basic Configuration on the Virtual Machine

Inside the VM, perform the following steps:

- Install cloud-init↗.
- Install the `qemu-guest-agent`.
- Install any required software.

Once installations are complete, run the following commands to clean cloud-init data and shut down the VM:

```
cloud-init clean
shutdown -h now
```

Step 2: Create a VM Snapshot

Using the KubeVirt Web Console:

1. Navigate to **Virtualization** > **Virtual Machines**.
2. Select the VM intended to serve as a template.
3. Click **Actions**, select **Create Snapshot**, name your snapshot, and confirm.

Step 3: Retrieve Disk Snapshot Resource Name

Obtain the complete snapshot resource name using one of these methods:

- **Via Web Console:**
  - Navigate to **Storage** > **Volume Snapshots**.
  - Find and record the full snapshot resource name under "Data Source."
- **Using kubectl:**

```
kubectl get volumesnapshots -n <NAMESPACE>
```

Record the complete snapshot resource name from the output.

Step 4: Create a DataSource Resource

Create the following DataSource resource in the `kube-public` namespace, ensuring you replace placeholders with the actual snapshot name and namespace:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataSource
metadata:
  annotations:
    cpaas.io/display-name: AlaudaOS-Clone
  labels:
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: linux
    virtualization.cpaas.io/storage-class: cephrbd
    virtualization.cpaas.io/access-mode: ReadWriteMany
    virtualization.cpaas.io/size: 30Gi
    virtualization.cpaas.io/volume-mode: Block
  name: alaudaos-clone
  namespace: kube-public
spec:
  source:
    snapshot:
      name: <Your Snapshot Resource Name>
      namespace: <Your Snapshot Namespace>
```

Label Parameters Explanation:

Key	Possible Values	Description
virtualization.cpaas.io/image-os-arch	amd64, arm64	VM OS architecture
virtualization.cpaas.io/image-os-type	linux, windows	VM OS type

Key	Possible Values	Description
virtualization.cpaas.io/storage-class	storage class name	Default storage class, adjustable during VM creation
virtualization.cpaas.io/access-mode	ReadWriteOnce, ReadWriteMany	Disk access mode; use ReadWriteMany for VM live migration
virtualization.cpaas.io/size	Capacity (Gi, Ti, etc.)	Default disk size; specify appropriate size
virtualization.cpaas.io/volume-mode	Block, Filesystem	Disk volume mode; Block mode recommended for better performance

Important:

- Ensure the namespace is `kube-public`.
- These disk-related parameters can be modified during VM creation but providing defaults simplifies the process.

Step 5: Create a New VM Using the Template

1. Access the KubeVirt Web Console, go to **Container Platform** > **Virtualization** > **Virtual Machines**.
2. Click **Create Virtual Machine**.
3. Under **Image**, select **Image Instance** as the Provision Method.
4. Select your newly created DataSource from the dropdown.
5. Configure any additional parameters as required and complete the VM creation process.

You have now successfully created and deployed new VMs using your VM template.

Troubleshooting

Pod Migration and Recovery from Abnormal Shutdown of Virtual Machine Nodes

TOC

Problem Description

Whether the node is **gracefully shut down** or experiences an **abnormal crash**, the virtual machine Pods running on that node will not automatically migrate to other healthy nodes.

Cause Analysis

The platform implements a virtual machine solution based on the open-source component KubeVirt. However, from the perspective of KubeVirt, it cannot differentiate between an actual

virtual machine crash and a connection failure caused by network or other issues. If virtual machines are migrated to other nodes indiscriminately, it may lead to multiple instances of the same virtual machine existing concurrently.

Solutions

When maintaining virtual machine nodes, manual actions are required according to this document. For both **graceful shutdown** and **abnormal crash** situations, virtual machine Pods must be manually evicted or forcibly deleted.

Note: The following commands must be executed on the Master node of the corresponding cluster.

Migration of Virtual Machine Pods during Graceful Shutdown

1. In the CLI tool, execute the following command to obtain node information. The `NAME` field in the returned information is the `Node-Name`.

   ```
   kubectl get nodes
   ```

   Output:

   ```
   NAME            STATUS   ROLES                  AGE   VERSION
   1.1.1.211   Ready    control-plane,master   99d   v1.28.8
   ```

2. (Optional) Execute the following command to view the virtual machine instances under the node.

   ```
   kubectl get vmis --all-namespaces -o wide | grep <Node-Name>  # Replace <Node-Name> in the command with the Node-Name obtained in step 1
   ```

   Output:

```
test-test          vm-t-export-clone   13d     Running      1.1.1.1   1.
1.1.211   True    False
```

3. Before the graceful shutdown, execute the following command to evict all virtual machine Pods on the node to be shut down. If the output appears as follows, it indicates that the eviction was successful.

```
kubectl drain <Node-Name> --delete-local-data --ignore-daemonsets=true
--force --pod-selector=kubevirt.io=virt-launcher   # Replace <Node-Name
> in the command with the Node-Name of the node to be shut down
```

   Output:

```
Flag --delete-local-data has been deprecated, This option is deprecated
and will be deleted. Use --delete-emptydir-data.
node/1.1.1.211 cordoned
evicting pod test-test/virt-launcher-vm-t-export-clone-hmnkk
pod/virt-launcher-vm-t-export-clone-hmnkk evicted
node/1.1.1.211 drained
```

4. After all virtual machines are started on other nodes, shut down the node.
5. After the node is shut down and rebooted, execute the following command to mark the node as schedulable.

```
kubectl uncordon <Node-Name> # Replace <Node-Name> in the command with
the Node-Name of the shut down and rebooted node
```

   Output:

```
node/1.1.1.211 uncordoned
```

6. At this point, the original virtual machine instances on that node have been migrated to other healthy nodes, and this node is now available for new Pod scheduling after rebooting.

Recovery from Abnormal Shutdown

1. In the CLI tool, execute the following command to obtain node information. The `NAME` field in the returned information is the `Node-Name`.

```
kubectl get nodes
```

Output:

```
NAME            STATUS   ROLES                  AGE   VERSION
1.1.1.211   Ready    control-plane,master   99d   v1.28.8
```

2. Execute the following command to forcibly delete all virtual machine Pods on the node.

```
kubectl get po -A -l kubevirt.io=virt-launcher -o wide | grep <Node-Name> | awk '{print "kubectl delete pod --force -n " $1, $2}'  | bash  # Replace <Node-Name> in the command with the Node-Name of the node that crashed abnormally.
```

3. Execute the following command to delete the volume attachments on that node.

```
kubectl get volumeattachments.storage.k8s.io | grep <Node-Name> | awk '{print $1}' | xargs kubectl delete volumeattachments.storage.k8s.io  # Replace <Node-Name> in the command with the Node-Name of the node that crashed abnormally.
```

4. Execute the following command to query if there are Pods with the label kubevirt.io=virt-api on the node that crashed abnormally.

```
kubectl -n kubevirt get po -l kubevirt.io=virt-api -o wide | grep <Node-Name> # Replace <Node-Name> in the command with the Node-Name of the node that crashed abnormally.
```

If they exist, execute the following command to delete the Pods.

```
kubectl -n kubevirt get po -l kubevirt.io=virt-api -o name | xargs kubectl -n kubevirt delete --force --grace-period=0
```

5. Execute the following command to query if there are Pods with the label kubevirt.io=virt-controller on the node that crashed abnormally.

```
kubectl -n kubevirt get po -l kubevirt.io=virt-controller -o wide | gre
p <Node-Name> # Replace <Node-Name> in the command with the Node-Name o
f the node that crashed abnormally.
```

   If they exist, execute the following command to delete the Pods.

```
kubectl -n kubevirt get po -l kubevirt.io=virt-controller -o name | xar
gs kubectl -n kubevirt delete --force --grace-period=0
```

6. At this point, the virtual machine instances will be migrated to other healthy nodes after the node experiences an abnormal shutdown.

Live Migration Error Messages and Solutions

Error Message	Cause	Solution
cannot migrate VMI which does not use masquerade, bridge with <annotation> VM annotation or a migratable plugin to connect to the pod network	The network configuration of the virtual machine does not support live migration.	Please check the following configurations: • Check the CNI network plugin used by the current cluster; Kube-OVN is recommended. • Check whether the "kubevirt.io/allow-pod-bridge-network-live-migration": "true" annotation exists in the `metadata.annotations` and `spec.template.metadata.annotations` fields of the corresponding YAML file of the virtual machine; if not, please add it manually.
• cannot migrate VMI: Unable to determine if PVC <pvc name> is shared, live migration requires that all PVCs must be shared (using ReadWriteMany access mode)	The storage type of the virtual machine does not support multi-node read-write (RWX) access mode.	The parameters related to the virtual machine cannot be modified after creation. Therefore, please recreate the virtual machine and select a storage type that supports multi-node read-write (RWX); CephRBD block storage is recommended. If issues persist after recreation, please contact the relevant personnel for assistance.

Error Message	Cause	Solution
• cannot migrate VMI: PVC <pvc name> is not shared, live migration requires that all PVCs must be shared (using ReadWriteMany access mode) • cannot migrate VMI: Backend storage PVC is not RWX • cannot migrate VMI with non-shared HostDisk		
Other error messages	The virtual machine does not support live migration.	Please contact the relevant personnel for assistance.

Troubleshooting Boot and Image Import

TOC

Reading the guest boot log

When a virtual machine fails to boot (kernel panic, missing root disk, stuck at the bootloader), the graphical and serial consoles may not show enough. Enable serial-console logging so the guest's boot output is captured to the launcher pod's logs:

```
spec:
  template:
    spec:
      domain:
        devices:
          logSerialConsole: true
```

After the virtual machine restarts, the launcher pod gains a `guest-console-log` container. Read the captured boot output with:

```
kubectl logs <virt-launcher-pod> -c guest-console-log -n <namespace>
```

Diagnosing a stuck image import

A bootable volume or virtual machine disk that stays in `Importing` usually has a `DataVolume` problem. Inspect the DataVolume's phase and conditions:

```
kubectl get datavolume <name> -n <ns> \
  -o jsonpath='phase={.status.phase} {range .status.conditions[*]}{.type}
={.status} {end}{"\n"}'
# A healthy completed import: phase=Succeeded Bound=True Ready=True Runni
ng=False

kubectl describe datavolume <name> -n <ns>   # events show pull / auth /
space errors
```

The `Bound` condition reflects the PVC, `Running` reflects the importer pod, and `Ready` indicates the disk is usable. A `Running=False` with a message points at the importer pod's failure — image not found, registry authentication, or insufficient space.

Capturing a virtual machine memory dump

To analyze a hung or crashed (but still running) virtual machine, capture its guest RAM to a PVC for offline analysis:

```
virtctl memory-dump get web-01 --claim-name=web-01-memory -n demo
```

The dump is written to the named PVC, which you can then download and inspect. See Using the virtctl Command-Line Tool.

Network

Introduction

Introduction

Advantages

Guides

Configure Network

Configure IP

Connect to the virtual machine directly via IP

Add Service

How To

Control Virtual Machine Networ

Procedure

Configuring SR-IOV

Terminology

Configuring Support

Introduction

ACP Virtualization With KubeVirt is deeply integrated with Kube-OVN, extending support for traditional virtual machine (VM) networking requirements and optimizing performance for specific scenarios.

TOC

Advantages

- IPv6 Support
  Full IPv6 Support.
- Static IP Retention
  Ensures VMs retain the same IP address after restarts, aligning with legacy VM usage patterns.
- Multi-Network Mode Support
  Supports multiple network modes such as container networks and SR-IOV, catering to diverse user scenarios.

Guides

Configure Network

Configure IP

Connect to the virtual machine directly via IP

Add Service

Configure Network

TOC

Configure IP

Refs to Configure IP

Connect to the virtual machine directly via IP

Refs to Preparing Kube-OVN Underlay Physical Network

Add Service

Refs to Add Service

Practical Guide

Control Virtual Machine Networ

- Procedure
- Using the API
- Result Verification

Configuring SR-IOV

- Terminology
- Constraints and Limitations
- Prerequisites
- Procedures
- Result Verification
- Related Notes

Configuring Support

- Prerequisites
- Procedure

Configuring

- Prerequisites
- Procedure

Control Virtual Machine Network Requests Through Network Policy

The platform's virtual machine solution is implemented based on the open-source component KubeVirt, which actually runs within Pods. By utilizing the functionality of Network Policies, it is possible to control the incoming and outgoing requests of virtual machines.

TOC

Procedure

1. Enter **Container Platform**.
2. In the left navigation bar, click **Network** > **Network Policies**.
3. Click **Create Network Policy**.
4. Configure the following parameters as needed.

Parameter	Description
Association Method	• **Compute Component**: Select the target compute component as needed; it is recommended to select **All** as the target compute component. • **Label Selector**: Match the Pods based on their labels.
Direction	• **Ingress**: Requests sent from the external to the Pod. • **Egress**: Requests sent from the Pod to the external; select this option if prohibiting the virtual machine from requesting a certain external address.
Protocol	Choose between TCP or UDP. **Note**: • When using domain names in the virtual machine to request external services, it is necessary to add a UDP protocol whitelist because DNS protocol uses UDP. • The form does not support configuring the ICMP protocol; once the whitelist rules are enabled, ICMP protocol will be disabled, which will result in the inability to perform Ping operations.
Access Ports	Specify which ports' traffic can be ingress or egress. If this field is left empty, traffic through all ports will be allowed by default. **Note**: It is necessary to allow ports 1053 and 53 for both UDP and TCP protocols here to permit DNS traffic egress; otherwise, domain name resolution will fail.
Remote Type	Specify the allowed remote types for access. Options include: compute component, namespace, and IP segments.

Parameter	Description
Exclude Remote	When the remote type is **IP Segment**, remove the specified IP from the whitelist (i.e., prohibit access). Single IP can be removed when input as `IP/32`. **Note**: This field only supports inputting IPs; if the corresponding IP of a domain name is unclear, use the command `curl -vvv <domain>` to request the domain and obtain the corresponding IP address from the returned information.

5. Click **Create**.

Using the API

The console form maps to a standard Kubernetes `NetworkPolicy` (`networking.k8s.io/v1`). The example below allows DNS egress and all outbound IP traffic **except** one address — the same outcome as the **Result Verification** scenario below:

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: vm-egress-allow
  namespace: demo
spec:
  # Association Method = Label Selector: select the virtual machine's pod
s.
  podSelector:
    matchLabels:
      vm.kubevirt.io/name: web-01
  policyTypes:
    - Egress
  egress:
    # Allow DNS (both UDP and TCP, ports 53 and 1053) or name resolution
fails.
    - to:
        - namespaceSelector: {}
      ports:
        - { protocol: UDP, port: 53 }
        - { protocol: TCP, port: 53 }
        - { protocol: UDP, port: 1053 }
        - { protocol: TCP, port: 1053 }
    # Remote Type = IP Segment (cidr) with an excluded address (Exclude R
emote).
    - to:
        - ipBlock:
            cidr: 0.0.0.0/0
            except:
              - 93.184.215.14/32
```

Field mapping: **Association Method (Label Selector)** → `spec.podSelector` (select the virtual machine's pods by `vm.kubevirt.io/name`); **Direction** → `policyTypes` plus `ingress` / `egress`; **Protocol / Access Ports** → `ports`; **Remote Type = IP Segment** → `ipBlock.cidr`; **Exclude Remote** → `ipBlock.except`.

Result Verification

This document verifies the setup using a virtual machine to access www.example.com↗.

Step One: Create a Virtual Machine and Network Policy Allowing All Traffic Through

1. Create the virtual machine, please refer to Create Virtual Machine for detailed steps.
2. Configure the network policy in the command namespace of the virtual machine, adding whitelist rules for both TCP and UDP protocols, with the following parameters:

   - Whitelist for TCP Protocol:

Parameter	Description
Association Method	Select **Compute Component**.
Target Compute Component	Select **All**.
Direction	Select **Egress**.
Protocol	Select **TCP**.
Remote Type	Select **IP Segment**
Remote	Enter **0.0.0.0/0**, indicating that all traffic is allowed to egress.

   - Whitelist Rules for UDP Protocol:

Parameter	Description
Direction	Select **Egress**.
Protocol	Select **UDP**.
Remote Type	Select **IP Segment**
Remote	Enter **0.0.0.0/0**, indicating that all traffic is allowed to egress.

3. After the network policy is created, log in to the virtual machine and execute the following command to request www.example.com↗.

```
curl www.example.com
```

4. The request is successful.

Step Two: Update Network Policy to Remove www.example.com↗ from Whitelist

1. Execute the following command to obtain the IP address for www.example.com↗, resulting in the IP address 93.184.215.14.

```
curl -vvv www.example.com
```

2. Update the network policy created in Step One, with the following updated parameters:

Parameter	Description
Exclude Remote	In the TCP protocol whitelist rules, fill in the exclude remote parameter with 93.184.215.14/32, indicating that IP address 93.184.215.14 is removed from the whitelist.

3. After updating the network policy, log in to the virtual machine and execute the following command to request www.example.com↗.

```
curl www.example.com
```

4. The request times out, indicating that the exclude remote functionality is effective.

Configuring SR-IOV

By configuring the physical server nodes to support the creation of virtual machines with SR-IOV (Single Root I/O Virtualization) network cards, lower latency for virtual machines is achieved, along with support for standalone IPv6 as well as dual-stack IPv4/IPv6 functionality.

TOC

Terminology

Term	Definition
Multus CNI	Acts as middleware for other CNI plugins to enable Kubernetes to support multiple network interfaces for Pods.
SR-IOV	Allows virtualization of the physical NIC on a node, splitting it into multiple VFs for use by Pods or virtual machines, providing superior network performance.
VF	A virtual device created from a physical PCI device; VFs can be allocated directly to virtual machines or containers, resembling independent physical PCI devices, significantly improving I/O performance.

Constraints and Limitations

The SR-IOV feature relies on glibc and only supports glibc versions 2.34 and above. However, both Kylin V10 and CentOS 7.x operating systems do not support this version, and thus, SR-IOV functionality cannot be used on these two operating systems.

Prerequisites

Obtain the following charts and images and upload them to the image repository. This document uses the repository address `build-harbor.example.cn` as an example. For specific methods to obtain the charts and images, please contact the relevant personnel.

Chart

- `build-harbor.example.cn/example/chart-sriov-network-operator:v3.15.0`

Images

- `build-harbor.example.cn/3rdparty/sriov/sriov-network-operator:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-network-operator-config-daemon:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-cni:4.13`
- `build-harbor.example.cn/3rdparty/sriov/ib-sriov-cni:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-network-device-plugin:4.13`
- `build-harbor.example.cn/3rdparty/sriov/network-resources-injector:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-network-operator-webhook:4.13`
- `build-harbor.example.cn/3rdparty/kubectl:v3.15.1`

Procedures

Note: All commands mentioned below are executed in the terminal.

1 Enabling SR-IOV in the Physical Machine's BIOS

Before configuration, use the following command to check the motherboard information.

```
dmidecode -t 1
# dmidecode 3.3
Getting SMBIOS data from sysfs.
SMBIOS 2.7 present.

Handle 0x0100, DMI type 1, 27 bytes
System Information
    Product Name: PowerEdge R620
    Version: Not Specified
    Serial Number: 7SJNF62
    UUID: 4c4c4544-0053-4a10-804e-b7c04f463632
    Wake-up Type: Power Switch
    SKU Number: SKU=NotProvided;ModelName=PowerEdge R620
    Family: Not Specified
```

The operation to enable SR-IOV in the BIOS varies among server manufacturers. Please refer to the corresponding manufacturer's documentation. Generally, the steps are as follows:

1. Reboot the server.
2. When the brand logo is displayed on the screen during BIOS POST, press the F2 key to enter the system setup.
3. Click **Processor Settings** > **Virtualization Technology**, and change **Virtualization Technology** setting to `Enabled`.
4. Click **Settings** > **Integrated devices**, and change **SR-IOV Global Enable** setting to `Enabled`.
5. Save the configuration and reboot the server.

2 Enabling IOMMU

The operation to enable IOMMU may vary across different operating systems. Please refer to the corresponding operating system documentation. This document uses CentOS as an example.

1. Edit the `/etc/default/grub` file and add `intel_iommu=on iommu=pt` to the `GRUB_CMDLINE_LINUX` configuration item.

```
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=centos/root rhgb qu
iet intel_iommu=on iommu=pt"
```

2. Execute the following command to generate the `grub.cfg` file.

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

3. Reboot the server.
4. Execute the following command, and if the output shows `IOMMU enabled`, it indicates that the enabling is successful.

```
dmesg | grep -i iommu
```

3 Loading the VFIO Module in the System Kernel

1. Execute the following command to load the vfio-pci module.

```
modprobe vfio-pci
```

2. Once loaded, execute the following command. If the configuration information can be displayed normally, then the VFIO kernel module has been loaded successfully.

```
# For CentOS, execute the following command to check the VFIO load
ing status
lsmod | grep vfio
vfio_pci               41993  0
vfio_iommu_type1       22440  0
vfio                   32657  2 vfio_iommu_type1, vfio_pci
irqbypass              13503  2 kvm, vfio_pc


# For Ubuntu, execute the following command to check the VFIO load
ing status
cat /lib/modules/$(uname -r)/modules.builtin | grep vfio
kernel/drivers/vfio/vfio.ko
kernel/drivers/vfio/vfio_virqfd.ko
kernel/drivers/vfio/vfio_iommu_type1.ko
kernel/drivers/vfio/pci/vfio-pci-core.ko
kernel/drivers/vfio/pci/vfio-pci.ko
```

4 Creating VF Devices

1. Execute the following command to see the currently supported VF devices.

```
find /sys -name *vfs*

/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_numvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.0/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.0/sriov_numvfs
```

   The output information indicates as follows:

   - **0000:05:00 .1**: The PCI address of the SR-IOV physical NIC enp5s0f1.
   - **0000:05:00 .0**: The PCI address of the SR-IOV physical NIC enp5s0f0.
   - **sriov_totalvfs**: Number of supported VFs.
   - **sriov_numvfs**: Current number of VFs.

2. Execute the following command to get information on the physical machine's NIC.

```
ifconfig

enp5s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.66.213  netmask 255.255.255.0  broadcast 192.
168.66.255
        inet6 1066::192:168:66:213  prefixlen 112  scopeid 0x0<glo
bal>
        inet6 fe80::a236:9fff:fe29:6c00  prefixlen 64  scopeid 0x2
0<link>
        ether a0:36:9f:29:6c:00  txqueuelen 1000  (Ethernet)
        RX packets 13889  bytes 1075801 (1.0 MB)
        RX errors 0  dropped 1603  overruns 0  frame 0
        TX packets 5057  bytes 440807 (440.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp5s0f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::a236:9fff:fe29:6c02  prefixlen 64  scopeid 0x2
0<link>
        ether a0:36:9f:29:6c:02  txqueuelen 1000  (Ethernet)
        RX packets 1714  bytes 227506 (227.5 KB)
        RX errors 0  dropped 1604  overruns 0  frame 0
        TX packets 70  bytes 19241 (19.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3. Execute the command `ethtool -i <NIC name>` to obtain the corresponding physical NIC's PCI address, as shown below.

```
ethtool -i enp5s0f0
driver: ixgbe
version: 5.15.0-76-generic
firmware-version: 0x8000030d, 14.5.8
expansion-rom-version:
bus-info: 0000:05:00.0     ## The PCI address of the enp5s0f0 NIC
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: yes


ethtool -i enp5s0f1
driver: ixgbe
version: 5.15.0-76-generic
firmware-version: 0x8000030d, 14.5.8
expansion-rom-version:
bus-info: 0000:05:00.1    ## The PCI address of the enp5s0f1 NIC
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: yes
```

4. Execute the following command to create a VF. This document takes configuring the enp5s0f1 NIC as an example. If multiple NICs need to be virtualized, all of them need to be configured.

```
cat /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_totalv
fs   ## Check the number of supported VFs
63

echo 8 > /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_n
umvfs  ## Set the current number of VFs

cat /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_numvfs
## Check the current number of VFs
8
```

5. Execute the following command to check if the VFs were created successfully.

Note: You can see the configured 8 VF addresses, such as `05:10.1`. These VF addresses need to be supplemented with the **Domain Identifier**, resulting in the final format: `0000:05:10.1`.

```
lspci | grep Virtual
00:11.0 PCI bridge: Intel Corporation C600/X79 series chipset PCI
Express Virtual Root Port (rev 05)
05:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:10.5 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.1 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.3 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.5 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
```

5 Binding the VFIO Driver

1. Download the binding script, and execute the `python3 dpdk-devbind.py -b vfio-pci <VF address with domain identifier>` command to bind the 8 VFs of the enp5s0f1 NIC to the vfio-pci driver, as shown below.

```
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.1
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.3
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.5
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.7
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.1
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.3
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.5
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.7
```

2. After binding successfully, execute the following command to check the binding results. Look for the already bound VFs in the **Network devices using DPDK-compatible driver** area in the output result. Among them, the VF device ID is `10ed`.

```
python3 dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
0000:05:10.1 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:10.3 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:10.5 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:10.7 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.1 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.3 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.5 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.7 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf

Network devices using kernel driver
===================================
0000:01:00.0 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
1 drv=tg3 unused=vfio-pci
0000:01:00.1 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
2 drv=tg3 unused=vfio-pci
0000:02:00.0 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
3 drv=tg3 unused=vfio-pci
0000:02:00.1 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
4 drv=tg3 unused=vfio-pci
0000:05:00.0 'Ethernet 10G 2P X520 Adapter 154d' if=enp5s0f0 drv=i
xgbe unused=vfio-pci *Active*
0000:05:00.1 'Ethernet 10G 2P X520 Adapter 154d' if=enp5s0f1 drv=i
xgbe unused=vfio-pci

No 'Baseband' devices detected
==============================

No 'Crypto' devices detected
============================

No 'DMA' devices detected
```

```
=========================

No 'Eventdev' devices detected
==============================

No 'Mempool' devices detected
=============================

No 'Compress' devices detected
==============================

No 'Misc (rawdev)' devices detected
===================================

No 'Regex' devices detected
===========================
```

6 Deploying the Multus CNI Plugin

1. Go to **Administrator**.
2. In the left navigation bar, click **Cluster Management** > **Clusters**.
3. Click the name of the virtual machine cluster and switch to the **Plugins** tab.
   - Deploy the **Multus CNI** plugin.

7 Deploying the sriov-network-operator

Execute the following command to deploy the sriov-network-operator.

```
REGISTRY=<$registry>  # Replace the <$registry> part with the reposit
ory address where the sriov-network-operator image is located, for ex
ample: REGISTRY=build-harbor.example.cn
NICSELECTOR=["<nics>"] # Replace the <nics> part with the NIC names,
for example: NICSELECTOR=["ens802f1","ens802f2"], separate multiple w
ith commas
NUMVFS=<numVfs> # Replace the <numVfs> part with the number of VFs, f
or example: NUMVFS=8

cat <<EOF | kubectl create -f -
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: sriov-network-operator
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: "kube-system"
  source:
    charts:
    - name: <chartName> # Replace <chartName> with the actual chart p
ath, for example: name = example/chart-sriov-network-operator
      releaseName: sriov-network-operator
      targetRevision: v3.15.0
    repoURL: $REGISTRY
  timeout: 120
  values:
    global:
      registry:
        address: $REGISTRY
    networkNodePolicy:
      nicSelector: $NICSELECTOR
      numVfs: $NUMVFS
EOF
```

8 Setting Node Role Identifier Labels for Physical Nodes

Note: Before performing this operation, ensure that the Pod of the `sriov-network-operator` is running normally.

1. Go to **Administrator**.
2. In the left navigation bar, click **Cluster Management** > **Clusters**.
3. Click the cluster name and switch to the **Nodes** tab.
4. Click the physical node that supports SR-IOV ⋮ > **Update Node Labels**.
5. Set the node label as follows:
   - `node-role.kubernetes.io/worker: ""`
6. Click **Update**.

9 Checking if the Resources are Created Successfully

In the CLI tool, execute the command `kubectl -n cpaas-system get sriovnetworknodestates` to check if the `sriovnetworknodestates` resource has been created successfully. If you see similar output below, it indicates that creation was successful. If the resource creation fails, check if the Multus CNI plugin and sriov-network-operator have been deployed successfully.

```
kubectl -n cpaas-system get sriovnetworknodestates
NAME                     SYNC STATUS           AGE
192.168.254.88           Succeeded             5d22h
```

10 Setting SR-IOV Node Feature Labels for Physical Nodes

Note: Before performing this operation, ensure that the `sriovnetworknodestates` resource has been successfully created.

1. Go to **Administrator**.
2. In the left navigation bar, click **Cluster Management** > **Clusters**.
3. Click the cluster name and switch to the **Nodes** tab.
4. Click the physical node that supports SR-IOV ⋮ > **Update Node Labels**.

5. Set the node label as follows:

   - `feature.node.kubernetes.io/network-sriov.capable: "true"`

11 Checking NIC Device Support

1. Execute the command `lspci -n -s <VF address with domain identifier>` to obtain the current NIC device's vendor ID and device ID, as shown below.

   ```
   lspci -n -s 0000:05:00.1
   05:00.1 0200: 8086:154d (rev 01)
   ```

   The output indicates:

   - **8086**: Vendor ID.
   - **154d**: Device ID.

2. Execute the command `lspci -s <VF address with domain identifier> -vvv | grep Ethernet` to obtain the current NIC name, as shown below.

   ```
   lspci -s 0000:05:00.1 -vvv | grep Ethernet
   05:00.1 Ethernet controller: Intel Corporation Ethernet 10G 2P X52
   0 Adapter (rev 01)
   ```

3. In the cpaas-system namespace, locate the configuration file named `supported-nic-ids` with type ConfigMap, and check if the current NIC's configuration information is in the support list within its data section.
   Note: If the current NIC is not in the support list, you need to refer to Step 4 to add the NIC to the configuration file. If the current NIC is already in the support list, skip Step 4.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: supported-nic-ids
  namespace: cpaas-system
data:
  Broadcom_bnxt_BCM57414_2x25G: 14e4 16d7 16dc
  Broadcom_bnxt_BCM75508_2x100G: 14e4 1750 1806
  Intel_i40e_10G_X710_SFP: 8086 1572 154c
  Intel_i40e_25G_SFP28: 8086 158b 154c
  Intel_i40e_40G_XL710_QSFP: 8086 1583 154c
  Intel_i40e_X710_X557_AT_10G: 8086 1589 154c
  Intel_i40e_XXV710: 8086 158a 154c
  Intel_i40e_XXV710_N3000: 8086 0d58 154c
  Intel_ice_Columbiaville_E810: 8086 1591 1889
  Intel_ice_Columbiaville_E810-CQDA2_2CQDA2: 8086 1592 1889
  Intel_ice_Columbiaville_E810-XXVDA2: 8086 159b 1889
  Intel_ice_Columbiaville_E810-XXVDA4: 8086 1593 1889
```

4. Add the current NIC to the data section of the support list in the format `<NIC Name>: <Vendor ID> <Device ID> <VF Device ID>`, as shown below.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: supported-nic-ids
  namespace: cpaas-system
data:
  Broadcom_bnxt_BCM57414_2x25G: 14e4 16d7 16dc
  Broadcom_bnxt_BCM75508_2x100G: 14e4 1750 1806

  Intel_Corporation_X520: 8086 154d 10ed            ## Add new NIC
information

  Intel_i40e_10G_X710_SFP: 8086 1572 154c
  Intel_i40e_25G_SFP28: 8086 158b 154c
  Intel_i40e_40G_XL710_QSFP: 8086 1583 154c
  Intel_i40e_X710_X557_AT_10G: 8086 1589 154c
  Intel_i40e_XXV710: 8086 158a 154c
  Intel_i40e_XXV710_N3000: 8086 0d58 154c
  Intel_ice_Columbiaville_E810: 8086 1591 1889
  Intel_ice_Columbiaville_E810-CQDA2_2CQDA2: 8086 1592 1889
  Intel_ice_Columbiaville_E810-XXVDA2: 8086 159b 1889
  Intel_ice_Columbiaville_E810-XXVDA4: 8086 1593 1889
```

Parameter configuration explanation:

- **Intel_Corporation_X520**: The name of the NIC, which can be customized.
- **8086**: Vendor ID.
- **154d**: Device ID.
- **10ed**: VF Device ID, which can be found in the binding results.

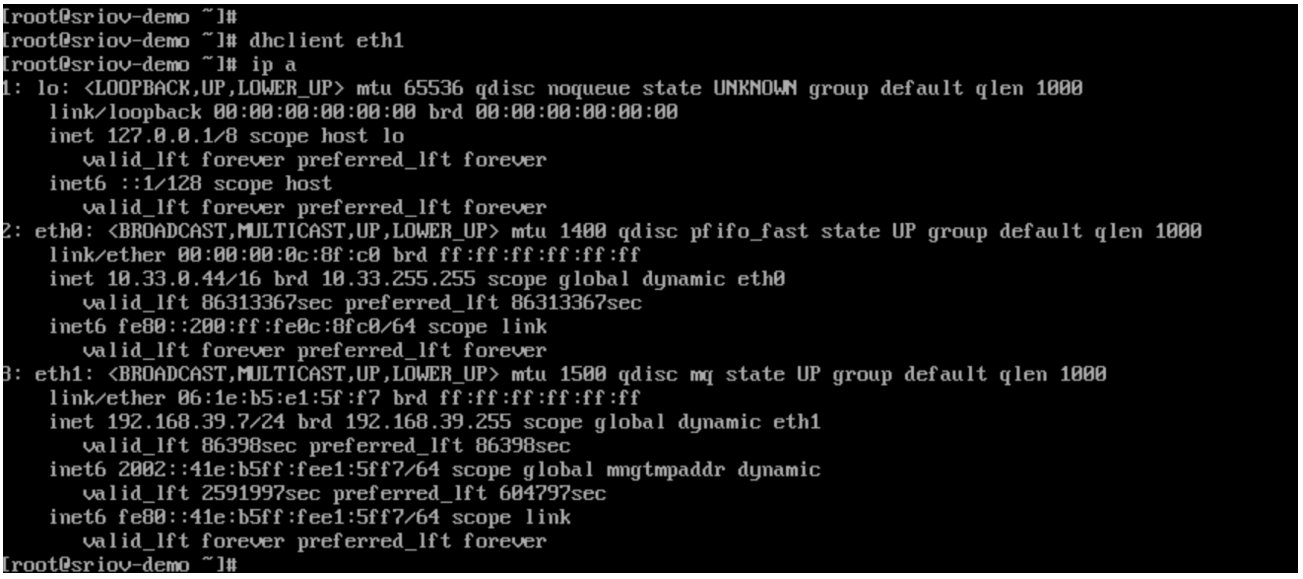
12 Configuring IP Address

Log in to the switch to configure DHCP (Dynamic Host Configuration Protocol).

Note: If it is not possible to use DHCP, please manually configure the IP address in the virtual machine.

Result Verification

1. Go to **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click **Create Virtual Machine**, and when adding an auxiliary network card, select **SR-IOV** as the **Network Type**.
4. Complete the creation of the virtual machine.
5. Access the virtual machine through VNC, you should see that eth1 has successfully obtained an IP address, indicating that the configuration has been successful.



Related Notes

Kernel Parameter Configuration for CentOS Virtual Machines

After the CentOS virtual machine uses the SR-IOV NIC, it is necessary to modify the kernel parameters for the corresponding NIC. The specific steps are as follows.

1. Open a terminal and execute the following command to modify the kernel parameters for the corresponding NIC. Replace the `<NIC Name>` part of the command with the actual NIC name.

```
sysctl -w net.ipv4.conf.<NIC Name>.rp_filter=2
echo "net.ipv4.conf.<NIC Name>.rp_filter=2" >> /etc/sysctl.conf
```

2. Execute the following command to load and apply all kernel parameter commands from the /etc/sysctl.conf file, so that the kernel configuration takes effect. When the value in the output information is 2, it indicates that the modification was successful.

```
sysctl -p
```

Output information:

```
net.ipv4.conf.<NIC Name>.rp_filter = 2
```

Configuring Virtual Machines to Use Network Binding Mode for IPv6 Support

The network binding mode is a plugin extension mechanism for virtual machine networking. By default, the platform uses a plugin called ManagedTap to enable IPv6 support for virtual machines. This plugin allows virtual machines to obtain IP addresses through the CNI's DHCP Server. Therefore, as long as the CNI's DHCP Server supports IPv6, virtual machines will also gain IPv6 capabilities.

Currently, we use Kube-OVN as the CNI. Since Kube-OVN's DHCP Server has full IPv6 support, virtual machines can achieve robust IPv6 functionality through the combination of ManagedTap and Kube-OVN.

TOC

Prerequisites

- ACP version must be v4.0.0 or higher.
- Kube-OVN is used as the CNI, and the virtual machine subnet is configured as Underlay.

Procedure

1 Add IPv6 Configuration to the Virtual Machine Subnet

```
kubectl edit subnet <subnet-name>
```

Add the following parameters under `spec`:

```
spec:
  enableDHCP: true
  enableIPv6RA: true
  u2oInterconnection: true
```

2 Create a Virtual Machine Using Network Binding Mode in the web console

When creating a virtual machine, select **Network Binding** as the network mode.

3 Access the Virtual Machine via VNC and Configure the Network Interface

For CentOS systems, edit the `/etc/sysconfig/network-scripts/ifcfg-enp1s0` file and add the following configuration:

```
IPV6INIT=yes
DHCPV6C=yes
IPV6_AUTOCONF=yes
```

restart network

```
systemctl restart network
```

4 Configure IPv6 Default Route

If the switch is configured to send Router Advertisement (RA) messages, manual route configuration is not required. The default route can be automatically learned through RA messages from the switch.

```
ip r r default via <subnet-v6-gateway>
```

Configuring Multi-NIC Virtual Machine

Use Kube-OVN together with Multus to provide multi-NIC support for virtual machines

TOC

Prerequisites

- Alauda Container Platform version must be v4.1.0 or higher.
- Kube-OVN is used as the CNI.
- Alauda Container Platform Networking for Multus is installed

Procedure

1 Create Secondary Network

1. Create NetworkAttachmentDefinition

   Execute the following command on the cluster control node:

Command

```
cat << EOF | kubectl create -f -
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: <name>
  namespace: <namespace>
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "kube-ovn",
    "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
    "provider": "<provider>"
    }'
EOF
```

Example

```
cat << EOF | kubectl create -f -
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: attachnet
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "kube-ovn",
    "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
    "provider": "attachnet.default.ovn"
    }'
EOF
```

Parameters:

- name: The name of NetworkAttachmentDefinition.
- namespace: The namespace of NetworkAttachmentDefinition, Must use the same namespace as the virtual machine.
- provider: The `<name>.<namespace>.ovn` of the current NetworkAttachmentDefinition. Kube-OVN will use this information to find the corresponding Subnet resource. Note that the suffix must be set to ovn.

2. Create a Kube-OVN Subnet

   If using Kube-OVN as a secondary network interface, the provider should be set to the corresponding NetworkAttachmentDefinition's `<name>.<namespace>.ovn`, and must end with the ovn suffix.

   Execute the following command on the cluster control node:

   Command

```
cat << EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: <name>
spec:
  protocol: IPv4
  enableDHCP: true
  provider: <provider>
  cidrBlock: <cidrBlock>
  gateway: <gateway>
  excludeIps:
  - <excludeIps>
EOF
```

   Example

```
cat << EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: attachnet
spec:
  protocol: IPv4
  enableDHCP: true
  provider: attachnet.default.ovn
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
  - 172.17.0.0..172.17.0.10
EOF
```

Parameters:

- name: The name of subnet.
- provider: The provider of NetworkAttachmentDefinition.
- cidrBlock: The subnet cidr.
- gateway: The gateway address.
- excludeIps: The set reserved IP will not be automatically allocated. For example, it can be used as the IP address for computing components' fixed IP.

2 Create a Multi-NIC Virtual Machine

1. Create Virtual Machine via UI
2. Switch to **YAML view** and add another nic to virtual machine
   Add new interface under `spec.template.spec.domain.devices.interfaces`
   Add new network under `spec.template.spec.networks`

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - bridge: {}
            name: default
          # new interface
          - bridge: {}
            name: dyniface1
      networks:
      - name: default
        pod: {}
      # new network
      - multus:
          networkName: <networkName>
        name: dyniface1
```

The networkName is the name of NetworkAttachmentDefinition.

3 Configure Network For New NIC

After the virtual machine starts, you need to enter the virtual machine and manually configure the network for the newly added nic.

4 Hotplug Network Interfaces

Hotplugging and unplugging network interfaces into a running Virtual Machine is supported

Hotplug is supported for interfaces using the virtio model connected through bridge binding or SR-IOV binding.

Hot-unplug is supported only for interfaces connected through bridge binding.

1. Adding an interface to a running VM
   Use kubectl edit to modify the virtual machine's YAML configuration

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - bridge: {}
            name: default
          # new interface
          - bridge: {}
            name: dyniface1
      networks:
      - name: default
        pod: {}
      # new network
      - multus:
          networkName: <networkName>
        name: dyniface1
```

2. Removing an interface from a running VM

   Use kubectl edit to modify the virtual machine's YAML configuration

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - bridge: {}
            name: default
          # set the interface state to absent
          - bridge: {}
            name: dyniface1
            state: absent
      networks:
      - name: default
        pod: {}
      # new network
      - multus:
          networkName: <networkName>
        name: dyniface1
```

Storage

Introduction

Introduction

Advantages

Guides

Managing Virtual Disks

Creating a Virtual Disk

Mounting a Virtual Disk

Expanding a Virtual Disk

Unmounting a Virtual Disk

Deleting a Virtual Disk

StorageProfile and cross-namespace clone scope

Introduction

ACP Virtualization with KubeVirt Storage provides persistent storage capabilities for virtual machines (VMs) by seamlessly integrating with Kubernetes-native storage mechanisms. It leverages **PersistentVolumeClaim** (PVC) to store VM disk data and utilizes the **Container Storage Interface** (CSI) to integrate with various storage systems. Additionally, the **Containerized Data Importer** (CDI) is employed to initialize VM disk data. Building on these foundations, the platform extends advanced functionalities for VM disk management, enabling comprehensive lifecycle control.

TOC

Advantages

- User-Friendly Operations
  Most VM disk operations can be easily performed via the **Web UI**, minimizing the need for CLI expertise.
- VM Disk Lifecycle Management
  Configure whether VM disks should be automatically deleted when the associated VM is terminated.

Guides

Managing Virtual Disks

Managing Virtual Disks

Data disks can be used to meet the data persistence requirements of the business.

A data disk is backed by a CDI `DataVolume` (which provisions a PVC of the same name). The operations below map to: create a `DataVolume`, hot-plug it to a virtual machine (`addvolume`), expand its PVC, detach it (`removevolume`), and delete the `DataVolume`.

TOC

StorageProfile and cross-namespace clone scope

Creating a Virtual Disk

Create a **data disk** for the virtual machine. Only **one** virtual disk can be added at a time; if multiple disks are needed, please repeat this operation.

Note: Virtual disks can be mounted online when the virtual machine is in **running** state.

Procedures

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Disk**.
3. Click on **Create Virtual Disk**.
4. Configure the information based on the following instructions.

Parameter	Description
Volume Mode	- **File System**: Mount the disk in a way that mounts the file directory. - **Block Device**: Mount the disk as a block device.
Storage Class	The platform maintains virtual machine disks by automatically creating and managing persistent volume claims. You need to specify the storage class required for dynamically creating persistent volume claims. Different storage classes support different volume modes. If there are no available storage classes for the selected volume mode, please contact the administrator for addition.
Delete with VM	If enabled, the disk data will also be deleted when the virtual machine is deleted.
Mount	- **Do Not Mount**: Only create the virtual disk; it can be mounted later when needed.

Parameter	Description
	- **Mount to VM**: Select the target virtual machine to which the virtual disk needs to be mounted.

5. Click on **Create**.

Using the API

Create a blank data disk as a `DataVolume`:

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: datadisk-1
  namespace: demo
spec:
  source:
    blank: {}
  storage:
    resources:
      requests:
        storage: 10Gi
    storageClassName: rbd-1
    volumeMode: Block
    accessModes:
      - ReadWriteMany
```

```bash
kubectl apply -f datadisk.yaml
# wait for the DataVolume to reach Succeeded and its PVC to be Bound
kubectl get datavolume datadisk-1 -n demo -w
```

Mounting a Virtual Disk

Mount the **data disk** to a virtual machine, attaching the already created virtual disk to the target virtual machine.

Note: Virtual disks can be mounted online when the virtual machine is in **running** state.

Procedures

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Disk**.
3. Click ⋮ > **Mount** next to the virtual disk to be mounted.
4. Select the target virtual machine and click **Mount**.

Using the API

Hot-plug the disk into a running virtual machine. With `virtctl`, use `--persist` so the disk is added to the virtual machine spec and survives a restart:

```
virtctl addvolume web-01 --volume-name=datadisk-1 --persist -n demo
```

This issues the `addvolume` subresource request (`AddVolumeOptions`). The disk then appears in `VirtualMachineInstance.status.volumeStatus` and progresses from `AttachedToNode` to `Ready`. Without `--persist`, the hot-plug applies to the running instance only and does not survive a restart.

Expanding a Virtual Disk

Expand the **system disk** and **data disk** already mounted to the virtual machine.

Procedures

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Machine**.
3. Click the name of the virtual machine to enter the **Details** page.
4. In the **Virtual Disk** area, find the disk to be expanded and click ⋮ > **Expand**.
5. Enter the new capacity and click **Expand**.

Using the API

Patch the PVC's requested size (the StorageClass must allow volume expansion):

```
kubectl patch pvc datadisk-1 -n demo --type merge \
  -p '{"spec":{"resources":{"requests":{"storage":"20Gi"}}}}'
```

`status.capacity.storage` grows to the new size; CSI drivers such as Ceph RBD expand online (no restart needed).

Unmounting a Virtual Disk

Unmount the **data disk** from the virtual machine; only virtual machines in the **stopped** state can unmount disks.

Procedures

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Disk**.
3. Click ⋮ > **Unmount** next to the virtual disk to be unmounted and confirm.

Using the API

Detach the hot-plugged disk:

```
virtctl removevolume web-01 --volume-name=datadisk-1 -n demo
```

This issues the `removevolume` subresource and the disk is detached (the entry leaves `VirtualMachineInstance.status.volumeStatus`). The KubeVirt API supports detaching from a **running** virtual machine, but make sure the disk is no longer in use inside the guest (unmounted at the OS level) before removing it, to avoid data loss.

Deleting a Virtual Disk

Deletion is only supported when the virtual disk is in an unmounted state.

Note: System disks cannot be deleted.

Procedures

1. Access the **Container Platform**.
2. In the left navigation bar, click on **Virtualization** > **Virtual Disk**.
3. Click ⋮ > **Delete** next to the virtual disk to be deleted and confirm.

Using the API

```
kubectl delete datavolume datadisk-1 -n demo
```

The backing PVC is owned by the `DataVolume` and is garbage-collected with it. If CDI already garbage-collected the `DataVolume`, delete the PVC directly (`kubectl delete pvc datadisk-1 -n demo`).

StorageProfile and cross-namespace clone scope

CDI publishes a `StorageProfile` for each StorageClass that describes how disks can be provisioned and cloned on that backend. Inspect it to see the supported access/volume modes and the clone method:

```
kubectl get storageprofile rbd-1 -o jsonpath='cloneStrategy={.status.clon
eStrategy} claimPropertySets={.status.claimPropertySets}{"\n"}'
# cloneStrategy=csi-clone claimPropertySets=[{RWX,Block},{RWO,Block},{RW
O,Filesystem}]
```

`cloneStrategy` (`csi-clone` , `snapshot` , or host-assisted `copy`) determines how a bootable volume or VM clone copies a disk; whether snapshot/clone work at all depends on

the StorageClass having these capabilities.

Cross-namespace clone scope: when creating a virtual machine or a bootable volume, the console only offers boot images from the **current namespace and** `kube-public` as clone sources. This UI guardrail — not RBAC — is the supported boundary on this platform: publish shared golden images to `kube-public`, where the built-in `os-images.kubevirt.io:view` role makes them clone-able by every authenticated user.

> CDI's upstream `datavolumes/source` permission (the `datavolume-cloner` ClusterRole) does **not** reliably gate CSI-path clones, so binding it is **not** a supported or sufficient way to expose images from an arbitrary non-public namespace. Use `kube-public` for shared images instead.

Backup and Recovery

Introduction

Introduction

Application Scenarios

Usage Limitations

Guides

Using Snapshots

Prerequisites

Notes

Creating a Snapshot

Rolling Back a Snapshot

Deleting a Snapshot

Using Velero

Prerequisites

Operation Steps

Introduction

ACP Virtualization With Kubevirt provides virtual machine snapshot capabilities, allowing users to back up and restore VMs via snapshots.

TOC

Application Scenarios

- Disaster Recovery & Failure Rollback
  When a virtual machine experiences data loss due to hardware failures, human errors (e.g., accidental file deletion), or malicious attacks (e.g., ransomware), snapshots serve as the last line of defense to restore operations.

Usage Limitations

- Snapshots can be created while the virtual machine is running (online) or stopped. For an application-consistent snapshot, install the QEMU guest agent in the guest; otherwise an online snapshot is crash-consistent. **Rolling back** a snapshot requires stopping the virtual machine first.

- The virtual machine disk must use a storage class that supports volume snapshots (a `VolumeSnapshotClass` is bound to it).

Guides

Using Snapshots

Using Velero

Using Snapshots

A virtual machine snapshot saves the current state of the virtual machine, and can be used to restore the virtual machine to that state in the event of an unexpected failure.

TOC

Prerequisites

- The **Volume Snapshot** has been deployed by the administrator in the platform management.
- Virtual machine snapshots are based on volume snapshots. Ensure that at least one disk is bound to a storage class that supports volume snapshots, such as CephFS built-in storage.
- Snapshots can be created while the virtual machine is **running** (online) or stopped (offline). For an **application-consistent** snapshot, install the QEMU guest agent in the guest so the filesystems can be quiesced; without it, an online snapshot is **crash-consistent** (the snapshot's `status.indications` reports `Online` and `NoGuestAgent`). **Rolling back** a snapshot requires the virtual machine to be **stopped** first — see stop the virtual machine.

Notes

If there are multiple storage types of the same kind in the cluster, for example, attaching multiple different sources of Ceph RBD storage, the disk snapshot functionality may not work properly when the virtual machine is using such storage.

Snapshot consistency

Check the snapshot's `status.indications` to know how trustworthy it is:

- `GuestAgent` — the QEMU guest agent froze the guest filesystems before the snapshot, so it is **application-consistent** (the safest to restore from).
- `NoGuestAgent` — no guest agent was available, so an online snapshot is **crash-consistent** (equivalent to pulling the power and rebooting). Install the guest agent for application-consistent snapshots.
- `QuiesceFailed` — the freeze was attempted but failed; treat the snapshot as crash-consistent.
- `Online` — the snapshot was taken while the virtual machine was running.

The persistent state of a virtual TPM (vTPM) is **not** captured in a snapshot, so a virtual machine that relies on a vTPM (for example for BitLocker) may need manual re-provisioning after a restore.

Creating a Snapshot

The contents included in a virtual machine snapshot: virtual machine settings and the state of the disks that support volume snapshots.

Procedures

1. Access **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Locate the virtual machine and click ⋮ > **Create Snapshot**.
4. Fill in the snapshot description. The description can help you document the current state of the virtual machine, such as `Initial Installation`, `Before Application Upgrade`.
5. Click **Create**. The time taken for the snapshot depends on network conditions and workload, please be patient.
6. Check the snapshot status.
   - When the snapshot changes to `Ready`, it indicates that the creation was successful.
   - If the snapshot remains in `Not Ready` status for a long time, click > View the reasons and troubleshoot, then recreate the snapshot.

Using the API

Create a `VirtualMachineSnapshot` that references the virtual machine:

```yaml
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineSnapshot
metadata:
  name: web-01-snap-1
  namespace: demo
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: web-01
```

```
kubectl apply -f snapshot.yaml
kubectl get virtualmachinesnapshot web-01-snap-1 -n demo \
  -o jsonpath='phase={.status.phase} readyToUse={.status.readyToUse} {.st
atus.indications}{"\n"}'
```

When `status.readyToUse` is `true`, the snapshot is complete. For a snapshot of a running virtual machine, `status.indications` includes `Online` (and `NoGuestAgent` when the guest agent is absent — the snapshot is then crash-consistent).

Rolling Back a Snapshot

Roll back the virtual machine settings and the disks that support volume snapshots to the state at the time the snapshot was created. For example, disks added after the snapshot creation will be removed; modified disk data will be restored.

Notes

If there are disks bound to a storage class that supports the LVM mechanism (for example, TopoLVM), please confirm with the administrator that the reclamation policy for that storage class is set to **Retain** (`reclaimPolicy: Retain`) to use the snapshot rollback feature correctly.

Procedures

1. Access **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click on ***Virtual Machine Name***.
4. In the **Snapshots** tab, locate the snapshot and click ⋮ > **Rollback**.
5. Read the prompt information on the interface, and click **Rollback** after confirming everything is correct.
   Note: The rollback operation cannot be aborted or undone, please proceed with caution.
6. Click on the snapshot name to check in the “Snapshot Rollback Records” if the rollback has been completed. The time required for the rollback depends on network conditions and

workload, please be patient.

Description

- If the rollback fails, the virtual machine state remains unchanged. You can start the virtual machine normally or attempt to roll back the snapshot again.
- If the virtual machine is started during the rollback process, it will revert to the state before it was stopped, and upon stopping the virtual machine again, it will continue rolling back to the state at the time of snapshot creation.
- To avoid operational conflicts, please ensure that the most recent rollback record has been completed before performing other operations on that virtual machine.

Using the API

Roll back by creating a `VirtualMachineRestore` that targets the virtual machine and references the snapshot. **Stop the target virtual machine first** — restore requires it to be stopped:

```yaml
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  name: web-01-restore-1
  namespace: demo
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: web-01
  virtualMachineSnapshotName: web-01-snap-1
```

```bash
kubectl apply -f restore.yaml
kubectl get virtualmachinerestore web-01-restore-1 -n demo \
  -o jsonpath='complete={.status.complete}{"\n"}'
```

The rollback is finished when `status.complete` is `true`.

Deleting a Snapshot

Delete unnecessary virtual machine snapshots to free up disk resources.

Notes

When deleting a rolled-back virtual machine snapshot, if the virtual machine disk needs to copy data based on the snapshot (for example, TopoLVM), you must wait until a virtual machine based on the rollback version has been started before deleting, otherwise the virtual machine will fail to start.

Procedures

1. Access **Container Platform**.
2. In the left navigation bar, click **Virtualization** > **Virtual Machines**.
3. Click on ***Virtual Machine Name***.
4. In the **Snapshots** tab, locate the target snapshot and click ⋮ > **Delete**.
5. Read the prompt information and click **Delete** after confirming everything is correct.

Using the API

```
kubectl delete virtualmachinesnapshot web-01-snap-1 -n demo
```

Using Velero

Velero is an open-source Kubernetes cluster backup and migration tool by VMware. KubeVirt provides a Velero plugin to support virtual machine backup and restoration.

TOC

Prerequisites

- **Kubernetes Version**: 1.20 or higher (Velero requirement)
- **S3 Storage**: For Velero BackupStorageLocation, use platform-provided Ceph or MinIO object storage
- **Block Storage**: Platform-provided Ceph RBD

Operation Steps

Preparation

1. Deploy Velero on the ACP platform via **Platform Management** → **Cluster Management** → **Backup and Recovery** → **Backup Repository**. Use object storage details to create the backup repository.
2. Enable the CSI feature and add the KubeVirt plugin:

   ```
   kubectl edit deploy -n cpaas-system velero
   ```

   Add the following to the Velero deployment:

   ```
   containers:
   - args:
     - server
     - --uploader-type=restic
     - --namespace=cpaas-system
     - --features=EnableCSI
     command:
     - /velero
   ```

   Add to the initContainers section:

   ```
   initContainers:
   - image: registry.example.org/3rdparty/kubevirt/kubevirt-velero-plugin:
   v0.7.0
     imagePullPolicy: IfNotPresent
     name: velero-plugin-kubevirt
     resources: {}
     terminationMessagePath: /dev/termination-log
     terminationMessagePolicy: File
     volumeMounts:
     - mountPath: /target
       name: plugins
   ```

3. Verify the KubeVirt plugin installation:

```
POD=$(kubectl -n cpaas-system get pod -l app.kubernetes.io/name=velero
-o jsonpath='{.items[0].metadata.name}')
kubectl -n cpaas-system exec -ti "$POD" -- /velero get plugins | grep k
ubevirt
```

Example output:

```
kubevirt-velero-plugin/backup-datavolume-action              BackupIte
mAction
kubevirt-velero-plugin/backup-datavolume-action              BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachine-action          BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachine-action          BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachineinstance-action  BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachineinstance-action  BackupIte
mAction
kubevirt-velero-plugin/restore-pod-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-pod-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-pvc-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-pvc-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-vm-action                     RestoreIt
emAction
kubevirt-velero-plugin/restore-vm-action                     RestoreIt
emAction
kubevirt-velero-plugin/restore-vmi-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-vmi-action                    RestoreIt
emAction
```

4. Adjust node-agent to mount the host kubelet directory and enable privileged mode (required for data movement via `/var/lib/kubelet`):

```
kubectl edit ds -n cpaas-system node-agent
```

Update with:

```
securityContext:
  privileged: true
volumeMounts:
- mountPath: /host_pods
  mountPropagation: HostToContainer
  name: host-pods
- mountPath: /var/lib/kubelet/plugins
  mountPropagation: HostToContainer
  name: host-plugins
- mountPath: /scratch
  name: scratch
securityContext:
  runAsUser: 0
volumes:
- hostPath:
    path: /var/lib/kubelet/pods
    type: ""
  name: host-pods
- hostPath:
    path: /var/lib/kubelet/plugins
    type: ""
  name: host-plugins
- emptyDir: {}
  name: scratch
```

Backup

1. Create a virtual machine with multiple disks as needed.
2. Create a backup resource:

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    velero.io/resource-timeout: 10m0s
    velero.io/source-cluster-k8s-gitversion: v1.30.4
    velero.io/source-cluster-k8s-major-version: "1"
    velero.io/source-cluster-k8s-minor-version: "30"
  labels:
    velero.io/storage-location: default
  name: example-backup
  namespace: cpaas-system
spec:
  csiSnapshotTimeout: 10m0s
  defaultVolumesToFsBackup: false
  hooks: {}
  includedNamespaces:
  - example-namespace
  itemOperationTimeout: 4h0m0s
  metadata: {}
  snapshotMoveData: true
  storageLocation: default
  ttl: 720h0m0s
```

3. Wait for the backup to complete and verify:

```
POD=$(kubectl -n cpaas-system get pod -l app.kubernetes.io/name=velero
-o jsonpath='{.items[0].metadata.name}')
kubectl -n cpaas-system exec -ti "$POD" -- /velero backup describe exam
ple-backup --details
kubectl -n cpaas-system exec -ti "$POD" -- /velero backup get example-b
ackup
```

Example output:

```
NAME             STATUS                          ERRORS   WARNINGS   CREAT
ED                          EXPIRES   STORAGE LOCATION   SELECTOR
example-backup   WaitingForPluginOperations   0        0          2025-
01-25 14:14:08 +0000 UTC   29d       default            <none>
```

4. Check dataupload status for data movement:

```
kubectl get dataupload -n cpaas-system
```

Example output:

```
NAME                   STATUS      STARTED   BYTES DONE    TOTAL BYTES
STORAGE LOCATION   AGE   NODE
example-backup-fhc6b   Completed   11h       21474836480   21474836480
default            11h   192.168.254.66
example-backup-qwwzh   Completed   11h       21474836480   21474836480
default            11h   192.168.254.15
```

5. Verify backup completion:

```
POD=$(kubectl -n cpaas-system get pod -l app.kubernetes.io/name=velero
-o jsonpath='{.items[0].metadata.name}')
kubectl -n cpaas-system exec -ti "$POD" -- /velero backup get example-b
ackup
```

Example output:

```
NAME             STATUS      ERRORS   WARNINGS   CREATED
EXPIRES   STORAGE LOCATION   SELECTOR
example-backup   Completed   0        0          2025-01-25 14:14:08 +0
000 UTC   29d       default            <none>
```

6. Check the backup repository (e.g., MinIO) for directories:

```
mc ls <backup-repository>
```

Example output:

```
[2025-01-26 09:23:08 CST]     0B backups/
[2025-01-26 09:23:08 CST]     0B kopia/
[2025-01-26 09:23:08 CST]     0B restic/
```

Restore

1. Delete the original virtual machine.
2. Create a restore resource via **Platform Management** → **Cluster Management** → **Backup and Recovery** → **Restore Management** or manually:

```yaml
apiVersion: velero.io/v1
kind: Restore
metadata:
  annotations:
    cpaas.io/description: ""
  finalizers:
  - restores.velero.io/external-resources-finalizer
  name: example-restore
  namespace: cpaas-system
spec:
  backupName: example-backup
  excludedResources:
  - nodes
  - events
  - events.events.k8s.io
  - backups.velero.io
  - restores.velero.io
  - resticrepositories.velero.io
  - csinodes.storage.k8s.io
  - volumeattachments.storage.k8s.io
  - backuprepositories.velero.io
  hooks: {}
  includedNamespaces:
  - example-namespace
  itemOperationTimeout: 4h0m0s
  namespaceMapping: {}
```

3. Verify the restore:

```
kubectl exec -ti -n cpaas-system velero-5df7bb7598-ljjrn -- /velero res
tore get
```

Check datadownload status:

```
kubectl get datadownload -n cpaas-system
```

Example output:

```
NAME       STATUS      STARTED   BYTES DONE    TOTAL BYTES   STORAGE LO
CATION   AGE   NODE
example-restore-7lfcm   Completed   11m       21474836480   21474836480
default            15m   192.168.254.66
example-restore-cjcd8   Completed   15m       21474836480   21474836480
default            15m   192.168.254.66
```

Restore status:

```
NAME   BACKUP           STATUS      STARTED                         COM
PLETED                       ERRORS   WARNINGS   CREATED
SELECTOR
aa     example-backup   Completed   2025-01-26 01:53:14 +0000 UTC   202
5-01-26 02:01:24 +0000 UTC   0        2          2025-01-26 01:53:14 +0
000 UTC   <none>
```

4. Confirm the virtual machine is restored and operational.

Cross-Cluster Restore

1. Deploy Velero on the new cluster and complete the preparation steps.
2. Configure the same backup repository (same bucket and directory) as the original cluster. The backup resource will automatically appear.
3. Follow the restore steps above.

Restore to a Different Namespace

Add a `namespaceMapping` field to the restore spec:

```
spec:
  namespaceMapping:
    example-namespace: test-namespace
```

Restore to a Different Storage Class

1. Create a ConfigMap in the `cpaas-system` namespace before initiating the restore:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: change-storage-class-config
  namespace: cpaas-system
  labels:
    velero.io/plugin-config: ""
    velero.io/change-storage-class: RestoreItemAction
data:
  vm-cephrbd: vm-topolvm-a
```

2. Ensure storage capabilities (e.g., cross-node access, RWX) are consistent.

3. Note: RWX mode modification is not currently supported.