How to Create a VM

You can create one or more virtual machines from the **Virtual Machines** page.

:::note

Please refer to this page for creating Windows virtual machines.

:::

1. Choose the option to create either one or multiple VM instances.
2. Select the namespace of your VMs, only the `harvester-public` namespace is visible to all users.
3. The VM Name is a required field.
4. (Optional) VM template is optional, you can choose `iso-image`, `raw-image` or `windows-iso-image` template to speed up your VM instance creation.
5. On the **Basics** tab, configure the following settings:

- **CPU** and **Memory**: You can allocate a maximum of **254** vCPUs. If virtual machines are not expected to fully consume the allocated resources most of the time, you can use the `overcommit-config` setting to optimize physical resource allocation.
- **SSHKey**: Select SSH keys or upload new keys.

1. Select a custom VM image on the **Volumes** tab. The default disk will be the root disk. You can add more disks to the VM.
2. To configure networks, go to the **Networks** tab.
    1. The **Management Network** is added by default, you can remove it if the VLAN network is configured.
    2. You can also add additional networks to the VMs using VLAN networks. You may configure the VLAN networks on **Advanced > Networks** first.
3. (Optional) Set node affinity rules on the **Node Scheduling** tab.
4. (Optional) Set workload affinity rules on the **VM Scheduling** tab.
5. Advanced options such as run strategy, os type and cloud-init data are optional. You may configure these in the **Advanced Options** section when applicable.

create-vm

To create virtual machines using the Kubernetes API, create a `VirtualMachine` object.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: new-vm
  namespace: default
spec:
  runStrategy: RerunOnFailure
  template:
    spec:
      domain:
        cpu:
          cores: 2
          sockets: 1
          threads: 1
```

```yaml
        memory: "3996Mi"
        devices:
          disks: []
          interfaces:
            - name: default
              model: virtio
              masquerade: {}
        machine:
          type: q35
        resources:
          requests:
            cpu: "125m"
            memory: "2730Mi"
          limits:
            cpu: 2
            memory: "4Gi"
        networks:
          - name: default
            pod: {}
```

For more information, see the API reference.

To create a virtual machine using the Harvester Terraform Provider, define a `harvester_virtualmachine` resource block:

```
resource "harvester_virtualmachine" "opensuse154" {
  name                 = "opensuse154"
  namespace            = "default"
  restart_after_update = true

  cpu    = 2
  memory = "2Gi"

  run_strategy = "RerunOnFailure"
  hostname     = "opensuse154"
  machine_type = "q35"

  ssh_keys = [
    harvester_ssh_key.mysshkey.id
  ]

  network_interface {
    name           = "nic-1"
    network_name   = harvester_network.cluster-vlan1.id
    wait_for_lease = true
  }

  disk {
    name       = "rootdisk"
    type       = "disk"
    size       = "10Gi"
    bus        = "virtio"
    boot_order = 1

    image       = harvester_image.opensuse154.id
    auto_delete = true
  }

  cloudinit {
    user_data_secret_name    = harvester_cloudinit_secret.cloud-config-
opensuse154.name
    network_data_secret_name = harvester_cloudinit_secret.cloud-config-
opensuse154.name
  }
}
```

Volumes

You can add one or more additional volumes via the `Volumes` tab, by default the first disk will be the root disk, you can change the boot order by dragging and dropping volumes, or using the arrow buttons.

A disk can be made accessible via the following types:

type	description
disk	This type will expose the volume as an ordinary disk to the VM.
cd-rom	This type will expose the volume as a cd-rom drive to the VM. It is read-only by default.

A volume's StorageClass can be specified when adding a new empty volume; for other volumes (such as VM images), the `StorageClass` is defined during image creation. If you are using external storage, ensure that the correct **StorageClass** and **Volume Mode** are selected. For example, a volume with the **nfs-csi** StorageClass must use the **Filesystem** volume mode.

:::info important

When creating volumes from a VM image, ensure that the volume size is greater than or equal to the image size. The volume may become corrupted if the configured volume size is less than the size of the underlying image. This is particularly important for qcow2 images because the virtual size is typically greater than the physical size.

By default, Harvester sets the volume size to either 10 GiB or the virtual size of the VM image, whichever is greater.

:::

create-vm

Adding a container disk

A container disk is an ephemeral storage volume that can be assigned to any number of VMs and provides the ability to store and distribute VM disks in the container image registry. A container disk is:

- An ideal tool if you want to replicate a large number of VM workloads or inject machine drivers that do not require persistent data. Ephemeral volumes are designed for VMs that need more storage but don't care whether that data is stored persistently across VM restarts or only expect some read-only input data to be present in files, like configuration data or secret keys.
- Not a good solution for any workload that requires persistent root disks across VM restarts.

A container disk is added when creating a VM by providing a Docker image. When creating a VM, follow these steps:

1. Go to the **Volumes** tab.
2. Select **Add Container**. add-container-volume
3. Enter a **Name** for the container disk.
4. Choose a disk **Type**.
5. Add a **Docker Image**.
   - A disk image, with the format qcow2 or raw, must be placed into the `/disk` directory.
   - Raw and qcow2 formats are supported, but qcow2 is recommended in order to reduce the container image's size. If you use an unsupported image format, the VM will get stuck in a `Running` state.
   - A container disk also allows you to store disk images in the `/disk` directory. An example of creating such a container image can be found here.
6. Choose a **Bus** type.

   add-container-volume

Networks

You can choose to add both the `management network` or `VLAN network` to your VM instances via the `Networks` tab, the `management network` is optional if you have the VLAN network configured.

Network interfaces are configured through the `Type` field. They describe the properties of the virtual interfaces seen inside the guest OS:

type	description
bridge	Connect using a Linux bridge
masquerade	Connect using iptables rules to NAT the traffic

Management Network

A management network represents the default VM eth0 interface configured by the cluster network solution that is present in each VM.

By default, VMs are accessible through the management network within the cluster nodes.

Secondary Network

It is also possible to connect VMs using additional networks with Harvester's built-in VLAN networks.

In bridge VLAN, virtual machines are connected to the host network through a linux `bridge`. The network IPv4 address is delegated to the virtual machine via DHCPv4. The virtual machine should be configured to use DHCP to acquire IPv4 addresses.

Node Scheduling

`Node Scheduling` allows you to constrain which nodes your VMs can be scheduled on based on node labels.

See the [Kubernetes Node Affinity Documentation] for more details.

VM Scheduling

`VM Scheduling` allows you to constrain which nodes your VMs can be scheduled on based on the labels of workloads (VMs and Pods) already running on these nodes, instead of the node labels.

For instance, you can combine `Required` with `Affinity` to instruct the scheduler to place VMs from two services in the same zone, enhancing communication efficiency. Likewise, the use of `Preferred` with `Anti-Affinity` can help distribute VMs of a particular service across multiple zones for increased availability.

See the [Kubernetes Pod Affinity and Anti-Affinity Documentation] for more details.

Advanced Options

Run Strategy

Available as of v1.0.2

Prior to v1.0.2, Harvester used the `Running` (a boolean) field to determine if the VM instance should be running. However, a simple boolean value is not always sufficient to fully describe the user's desired behavior. For example, in some cases the user wants to be able to shut down the instance from inside the virtual machine. If the `running` field is used, the VM will be restarted immediately.

In order to meet the scenario requirements of more users, the `RunStrategy` field is introduced. This is mutually exclusive with `Running` because their conditions overlap somewhat. There are currently four `RunStrategies` defined:

- Always: The VM instance will always exist. If VM instance crashes, a new one will be spawned. This is the same behavior as `Running: true`.
- RerunOnFailure (default): If the previous instance failed in an error state, a VM instance will be respawned. If the guest is successfully stopped (e.g. shut down from inside the guest), it will not be recreated.
- Manual: The presence or absence of a VM instance is controlled only by the `start/stop/restart` VirtualMachine actions.
- Stop: There will be no VM instance. If the guest is already running, it will be stopped. This is the same behavior as `Running: false`.

Reserved Memory

Each VM is configured with a memory value, this memory is targeted for the VM guest OS to see and use. In Harvester, the VM is carried by a Kubernetes POD. The memory limitation is achieved by Kubernetes [Resource requests and limits of Pod and container]. Certain amount of memory is required to simulate and manage the `CPU/Memory/Storage/Network/...` for the VM to run. Harvester and KubeVirt summarize such additional memory as the VM `Memory Overhead`. The `Memory Overhead` is computed by a complex formula. However, sometimes the OOM(Out Of Memory) can still happen and the related VM is killed by the Harvester OS, the direct cause is that the whole POD/Container exceeds its memory limits. From practice, the `Memory Overhead` varies on different kinds of VM, different kinds of VM operating system, and also depends on the running workloads on the VM.

Harvester adds a `Reserved Memory` field and a setting `additional-guest-memory-overhead-ratio` for users to adjust the guest OS memory and the `Total Memory Overhead`.

The `Total Memory Overhead` = automatically computed `Memory Overhead` + Harvester `Reserved Memory`.

All the details are described in the setting additional-guest-memory-overhead-ratio.

:::important

Read the document carefully, understand how it works and set a proper value on this field.

:::

Cloud Configuration

Harvester supports the ability to assign a startup script to a virtual machine instance which is executed automatically when the VM initializes.

These scripts are commonly used to automate injection of users and SSH keys into VMs in order to provide remote access to the machine. For example, a startup script can be used to inject credentials into a VM that allows an Ansible job running on a remote host to access and provision the VM.

Cloud-init

Cloud-init is a widely adopted project and the industry standard multi-distribution method for cross-platform cloud instance initialization. It is supported across all major cloud image provider like SUSE, Redhat, Ubuntu and etc., cloud-init has established itself as the defacto method of providing startup scripts to VMs.

Harvester supports injecting your custom cloud-init startup scripts into a VM instance through the use of an ephemeral disk. VMs with the cloud-init package installed will detect the ephemeral disk and execute custom user-data and network-data scripts at boot.

Example of password configuration for the default user:

```
#cloud-config
password: password
chpasswd: { expire: False }
ssh_pwauth: True
```

Example of network-data configuration using DHCP:

```
network:
  version: 1
  config:
    - type: physical
      name: eth0
      subnets:
        - type: dhcp
    - type: physical
      name: eth1
      subnets:
        - type: dhcp
```

You can also use the `Advanced > Cloud Config Templates` feature to create a pre-defined cloud-init configuration template for the VM.

:::note

The network configuration of a virtual machine running an Ubuntu release later than 16.04 is likely managed by `netplan` by default. Please use the following `network` settings as reference for DHCP configuration to prevent IP address conflicts when restoring a virtual machine from a snapshot or backup.

```
network:
  ethernets:
    enp1s0:
      dhcp4: true
      dhcp6: true
      dhcp-identifier: mac
  version: 2
```

If dhcp-identifier: mac is not specified, the restored virtual machine will receive the same IP address from the DHCP server. This happens because netplan defaults to using the machine ID as the DHCP client identifier. As a result, the restored virtual machine will receive the same IP address from the DHCP server and lead to IP address conflicts.

:::

Installing the QEMU guest agent

The QEMU guest agent is a daemon that runs on the virtual machine instance and passes information to the host about the VM, users, file systems, and secondary networks.

`Install guest agent` checkbox is enabled by default when a new VM is created.

:::note

If your OS is openSUSE and the version is less than 15.3, please replace `qemu-guest-agent.service` with `qemu-ga.service`.

:::

TPM Device

Available as of v1.2.0

Trusted Platform Module (TPM) is a cryptoprocessor that secures hardware using cryptographic keys.

According to Windows 11 Requirements, the TPM 2.0 device is a hard requirement of Windows 11.

In the Harvester UI, you can add an emulated TPM 2.0 device to a VM by checking the `Enable TPM` box in the **Advanced Options** tab.

:::note

Currently, only non-persistent vTPMs are supported, and their state is erased after each VM shutdown. Therefore, Bitlocker should not be enabled.

:::

One-time Boot For ISO Installation

When creating a VM to boot from cd-rom, you can use the **bootOrder** option so that the OS can boot from cd-rom during image installation, and boot from the disk when the installation is complete without unmounting the cd-rom.

The following example describes how to install an ISO image using openSUSE Leap 15.4:

1. Click **Images** in the left sidebar and download the openSUSE Leap 15.4 ISO image.
2. Click **Virtual Machines** in the left sidebar, then create a VM. You need to fill up those VM basic configurations.
3. Click the **Volumes** tab, In the **Image** field, select the image downloaded in step 1 and ensure **Type** is `cd-rom`
4. Click **Add Volume** and select an existing **StorageClass**.
5. Drag **Volume** to the top of **Image Volume** as follows. In this way, the **bootOrder** of **Volume** will become `1`.

one-time-boot-create-vm-bootorder

6. Click **Create**.
7. Open the VM web-vnc you just created and follow the instructions given by the installer.
8. After the installation is complete, reboot the VM as instructed by the operating system (you can remove the installation media after booting the system).
9. After the VM reboots, it will automatically boot from the disk volume and start the operating system.