RKE1 and RKE2 clusters can be provisioned in Rancher using the built-in Harvester Node Driver. Harvester provides load balancer and Harvester cluster storage passthrough support to the guest Kubernetes cluster.

In this page we will learn:

- How to deploy the Harvester cloud provider in both RKE1 and RKE2 cluster.
- How to use the Harvester load balancer.

### Backward Compatibility Notice

:::note

Please note a known backward compatibility issue if you're using the Harvester cloud provider version **v0.2.2** or higher. If your Harvester version is below **v1.2.0** and you intend to use newer RKE2 versions (i.e., >= `v1.26.6+rke2r1`, `v1.25.11+rke2r1`, `v1.24.15+rke2r1`), it is essential to upgrade your Harvester cluster to v1.2.0 or a higher version before proceeding with the upgrade of the guest Kubernetes cluster or Harvester cloud provider.

For a detailed support matrix, please refer to the **Harvester CCM & CSI Driver with RKE2 Releases** section of the official website.

:::

## Deploying

### Prerequisites

- The Kubernetes cluster is built on top of Harvester virtual machines.
- The Harvester virtual machines run as guest Kubernetes nodes are in the same namespace.

:::info important

Each Harvester VM must have the `macvlan` kernel module, which is required for the `LoadBalancer` services of the **DHCP** IPAM mode.

To check if the kernel module is available, access the VM and run the following commands:

```
lsmod | grep macvlan
sudo modprobe macvlan
```

The kernel module is likely to be missing if the following occur:

- `$ lsmod | grep macvlan` does not produce output.
- `$ sudo modprobe macvlan` displays an error message similar to `modprobe: FATAL: Module macvlan not found in directory /lib/modules/5.14.21-150400.22-default`.

By default, the `macvlan` kernel module is not included in SUSE Linux Enterprise 15 Service Pack 4/5/6 minimal cloud images (see Issue #6418). Those images contain the `kernel-default-base` package, which includes only the base modules. However, the `macvlan` kernel driver becomes available when you install the `kernel-default` package.

To eliminate the need for manual intervention after the guest cluster is provisioned, build your own cloud images using the openSUSE Build Service (OBS). You must remove the `kernel-default-base` package and add the `kernel-default` package in the `Minimal.kiwi` file to ensure that the resulting cloud image includes the `macvlan` kernel module. For more information, see Custom SUSE VM Images.

:::

### Deploying to the RKE1 Cluster with Harvester Node Driver

:::caution

Rancher Kubernetes Engine (RKE) will reach the end of its life on **July 31, 2025**. Harvester **v1.6.0** and later versions will not support RKE. Switching to RKE2, which provides a more secure and efficient environment, is recommended.

In-place upgrades are not an option, so you must create new RKE2 clusters and migrate the workloads from your existing RKE clusters (known as replatforming). For more information, see RKE End of Life.

:::

When spinning up an RKE cluster using the Harvester node driver, you can perform two steps to deploy the `Harvester` cloud provider:

1. Select `Harvester(Out-of-tree)` option.

2. Install `Harvester Cloud Provider` from the Rancher marketplace.

### Deploying to the RKE2 Cluster with Harvester Node Driver

When spinning up an RKE2 cluster using the Harvester node driver, select the `Harvester` cloud provider. The node driver will then help deploy both the CSI driver and CCM automatically.

Starting with Rancher v2.9.0, you can configure a specific folder for cloud config data using the **Data directory configuration path** field.

### Manually Deploying to the RKE2 Cluster

1. Generate cloud config data using the script `generate_addon.sh`, and then place the data on every custom node (directory: `/etc/kubernetes/cloud-config`).

```
  curl -sfL https://raw.githubusercontent.com/harvester/cloud-provider-
harvester/master/deploy/generate_addon.sh | bash -s <serviceaccount name>
<namespace>
```

:::note

```
The script depends on `kubectl` and `jq` when operating the Harvester cluster, and
functions only when given access to the `Harvester Cluster` kubeconfig file.

You can find the `kubeconfig` file in one of the Harvester management nodes in the
```

````
`/etc/rancher/rke2/rke2.yaml` path. The server IP must be replaced with the VIP
address.
````

Example of content:

````
```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: <redacted>
    server: https://127.0.0.1:6443
  name: default
# ...
```

You must specify the namespace in which the guest cluster will be created.
````

:::

````
Example of output:

```yaml
########## cloud config ############
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: <CACERT>
    server: https://HARVESTER-ENDPOINT/k8s/clusters/local
  name: local
contexts:
- context:
    cluster: local
    namespace: default
    user: harvester-cloud-provider-default-local
  name: harvester-cloud-provider-default-local
current-context: harvester-cloud-provider-default-local
kind: Config
preferences: {}
users:
- name: harvester-cloud-provider-default-local
  user:
    token: <TOKEN>

########## cloud-init user data ############
write_files:
- encoding: b64
  content: <CONTENT>
  owner: root:root
  path: /etc/kubernetes/cloud-config
  permissions: '0644'
```
````

1. On the RKE2 cluster creation page, go to the **Cluster Configuration** screen and set the value of **Cloud Provider** to **External**.

2. Copy and paste the `cloud-init user data` content to **Machine Pools** > **Show Advanced** > **User Data**.

3. Add the `HelmChart` CRD for `harvester-cloud-provider` to **Cluster Configuration** > **Add-On Config** > **Additional Manifest**.

   You must replace `<cluster-name>` with the name of your cluster.

   ```
   apiVersion: helm.cattle.io/v1
   kind: HelmChart
   metadata:
     name: harvester-cloud-provider
     namespace: kube-system
   spec:
     targetNamespace: kube-system
     bootstrap: true
     repo: https://raw.githubusercontent.com/rancher/charts/dev-v2.9
     chart: harvester-cloud-provider
     version:  104.0.2+up0.2.6
     helmVersion: v3
     valuesContent: |-
       global:
         cattle:
           clusterName: <cluster-name>
   ```

4. To create the load balancer, add the annotation `cloudprovider.harvesterhci.io/ipam: <dhcp|pool>` .

### Deploying to the RKE2 custom cluster (experimental)

1. Generate cloud config data using the script `generate_addon.sh` , and then place the data on every custom node (directory: `/etc/kubernetes/cloud-config` ).

   ```
   curl -sfL https://raw.githubusercontent.com/harvester/cloud-provider-
   harvester/master/deploy/generate_addon.sh | bash -s <serviceaccount name>
   <namespace>
   ```

:::note

````
The script depends on `kubectl` and `jq` when operating the Harvester cluster, and
functions only when given access to the `Harvester Cluster` kubeconfig file.

You can find the `kubeconfig` file in one of the Harvester management nodes in the
`/etc/rancher/rke2/rke2.yaml` path. The server IP must be replaced with the VIP
address.
````

Example of content:

````
```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: <redacted>
    server: https://127.0.0.1:6443
  name: default
# ...
```

You must specify the namespace in which the guest cluster will be created.
````

:::

````
Example of output:

```yaml
########## cloud config ############
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: <CACERT>
    server: https://HARVESTER-ENDPOINT/k8s/clusters/local
  name: local
contexts:
- context:
    cluster: local
    namespace: default
    user: harvester-cloud-provider-default-local
  name: harvester-cloud-provider-default-local
current-context: harvester-cloud-provider-default-local
kind: Config
preferences: {}
users:
- name: harvester-cloud-provider-default-local
  user:
    token: <TOKEN>

########## cloud-init user data ############
write_files:
- encoding: b64
  content: <CONTENT>
  owner: root:root
````
````

```
  path: /etc/kubernetes/cloud-config
  permissions: '0644'
```

1. Create a VM in the Harvester cluster with the following settings:
    - **Basics** tab: The minimum requirements are 2 CPUs and 4 GiB of RAM. The required disk space depends on the VM image.
    - **Networks** tab: Specify a network name with the format `nic-<number>`.
    - **Advanced Options** tab: Copy and paste the content of the **Cloud Config User Data** screen.
2. On the **Basics** tab of the **Cluster Configuration** screen, select **Harvester** as the **Cloud Provider** and then select **Create** to spin up the cluster.
1. On the **Registration** tab, perform the steps required to run the RKE2 registration command on the VM.

### Deploying to the K3s cluster with Harvester node driver (experimental)

When spinning up a K3s cluster using the Harvester node driver, you can perform the following steps to deploy the harvester cloud provider:

1. Use `generate_addon.sh` to generate cloud config.

    ```
    curl -sfL https://raw.githubusercontent.com/harvester/cloud-provider-harvester/master/deploy/generate_addon.sh | bash -s <serviceaccount name> <namespace>
    ```

    The output will look as follows:

    ```
    ########## cloud config ############
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority-data: <CACERT>
        server: https://HARVESTER-ENDPOINT/k8s/clusters/local
      name: local
    contexts:
    - context:
    ```

```
    cluster: local
    namespace: default
    user: harvester-cloud-provider-default-local
  name: harvester-cloud-provider-default-local
current-context: harvester-cloud-provider-default-local
kind: Config
preferences: {}
users:
- name: harvester-cloud-provider-default-local
  user:
    token: <TOKEN>


########## cloud-init user data ############
write_files:
- encoding: b64
  content: <CONTENT>
  owner: root:root
  path: /etc/kubernetes/cloud-config
  permissions: '0644'
```

2. Copy and paste the `cloud-init user data` content to **Machine Pools >Show Advanced > User Data**.

3. Add the following `HelmChart` yaml of `harvester-cloud-provider` to **Cluster Configuration > Add-On Config > Additional Manifest**.

```
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: harvester-cloud-provider
  namespace: kube-system
spec:
  targetNamespace: kube-system
  bootstrap: true
  repo: https://charts.harvesterhci.io/
  chart: harvester-cloud-provider
  version: 0.2.2
  helmVersion: v3
```

4. Disable the `in-tree` cloud provider in the following ways:

   - Click the `Edit as YAML` button.

   - Disable `servicelb` and set `disable-cloud-controller: true` to disable the default K3s cloud controller.

```
machineGlobalConfig:
  disable:
    - servicelb
  disable-cloud-controller: true
```

- Add `cloud-provider=external` to use the Harvester cloud provider.

```
machineSelectorConfig:
  - config:
      kubelet-arg:
      - cloud-provider=external
      protect-kernel-defaults: false
```

With these settings in place a K3s cluster should provision successfully while using the external cloud provider.

## Upgrade Cloud Provider

### Upgrade RKE2

The cloud provider can be upgraded by upgrading the RKE2 version. You can upgrade the RKE2 cluster via the Rancher UI as follows:

1. Click ☰ **> Cluster Management**.
2. Find the guest cluster that you want to upgrade and select ⋮ **> Edit Config**.
3. Select **Kubernetes Version**.
4. Click **Save**.

### Upgrade RKE/K3s

RKE/K3s upgrade cloud provider via the Rancher UI, as follows:

1. Click ☰ **> RKE/K3s Cluster > Apps > Installed Apps**.
2. Find the cloud provider chart and select ⋮ **> Edit/Upgrade**.
3. Select **Version**.
4. Click **Next > Update**.

:::info

The upgrade process for a single-node guest cluster may stall when the new `harvester-cloud-provider` pod is stuck in the *Pending* state. This issue is caused by a section in the `harvester-cloud-provider` deployment that describes the rolling update strategy. Specifically, the default value conflicts with the `podAntiAffinity` configuration in single-node clusters.

For more information, see this GitHub issue comment. To address the issue, manually delete the old `harvester-cloud-provider` pod. You might need to do this multiple times until the new pod can be successfully scheduled.

:::

## Load Balancer Support

Once you've deployed the Harvester cloud provider, you can leverage the Kubernetes `LoadBalancer` service to expose a microservice within the guest cluster to the external world. Creating a Kubernetes `LoadBalancer` service assigns a dedicated Harvester load balancer to the service, and you can make adjustments through the `Add-on Config` within the Rancher UI.

### IPAM

Harvester's built-in load balancer offers both **DHCP** and **Pool** modes, and you can configure it by adding the annotation `cloudprovider.harvesterhci.io/ipam: $mode` to its corresponding service. Starting from Harvester cloud provider >= v0.2.0, it also introduces a unique **Share IP** mode. A service shares its load balancer IP with other services in this mode.

- **DCHP:** A DHCP server is required. The Harvester load balancer will request an IP address from the DHCP server.
- **Pool:** An IP pool must be configured first. The Harvester load balancer controller will allocate an IP for the load balancer service following the IP pool selection policy.
- **Share IP:** When creating a new load balancer service, you can re-utilize an existing load balancer service IP. The new service is referred to as a secondary service, while the currently chosen service is the primary one. To specify the primary service in the secondary service, you can add the annotation `cloudprovider.harvesterhci.io/primary-service: $primary-service-name`. However, there are two known limitations:
  - Services that share the same IP address can't use the same port.
  - Secondary services cannot share their IP with additional services.

:::note

Modifying the `IPAM` mode isn't allowed. You must create a new service if you intend to change the `IPAM` mode.

:::

## Health checks

Beginning with Harvester cloud provider v0.2.0, additional health checks of the `LoadBalancer` service within the guest Kubernetes cluster are no longer necessary. Instead, you can configure liveness and readiness probes for your workloads. Consequently, any unavailable pods will be automatically removed from the load balancer endpoints to achieve the same desired outcome.