Host KVM Installation

System Requirements for KVM Hypervisor Hosts

KVM is included with a variety of Linux-based operating systems. Although you are not required to run these distributions, the following are recommended:

- CentOS / RHEL: 7.X
- CentOS / RHEL / Binary-compatible variants: 8.X
- Ubuntu: 18.04 +
- openSUSE / SLES: 15.2 +

The main requirement for KVM hypervisors is the libvirt and Qemu version. No matter what Linux distribution you are using, make sure the following requirements are met:

- libvirt: 1.2.0 or higher
- Qemu/KVM: 1.5 or higher (2.5 or higher recommended)

The default bridge in CloudStack is the Linux native bridge implementation (bridge module). CloudStack includes an option to work with OpenVswitch, the requirements are listed below

- libvirt: 1.2.0 or higher
- openvswitch: 1.7.1 or higher

Not all versions of Qemu/KVM may support dynamic scaling of Instances. Some combinations may result CPU or memory related failures during Instance deployment.

In addition, the following hardware requirements apply:

- Within a single cluster, the hosts must be of the same distribution version.
- All hosts within a cluster must be homogeneous. The CPUs must be of the same type, count, and feature flags.
- Must support HVM (Intel-VT or AMD-V enabled)
- 64-bit x86 CPU (more cores results in better performance)
- 4 GB of memory
- At least 1 NIC
- When you deploy CloudStack, the hypervisor host must not have any Instances already running. These will be destroy by CloudStack.

KVM Installation Overview

If you want to use the Linux Kernel Virtual Machine (KVM) hypervisor to run Guest Instances, install KVM on the host(s) in your cloud. The material in this section doesn't duplicate KVM installation docs. It provides the CloudStack-specific steps that are needed to prepare a KVM host to work with CloudStack.

Warning

Before continuing, make sure that you have applied the latest updates to your host.

Warning

It is NOT recommended to run services on this host not controlled by CloudStack.

Warning

Certain servers such as Dell provide the option to choose the Power Management Profile. The Active Power Controller enables Dell System DBPM (Demand Based Power Management) which can restrict the visibility of the maximum CPU clock speed available to the OS, which in turn can lead

to CloudStack fetching the incorrect CPU speed of the server. To ensure that CloudStack can always fetch the maximum cpu speed on the server, ensure that “OS Control” is set as the Power Management Profile.

The procedure for installing a KVM Hypervisor Host is:

1. Prepare the Operating System
2. Install and configure libvirt
3. Configure Security Policies (AppArmor and SELinux)
4. Install and configure the Agent

Prepare the Operating System

The OS of the Host must be prepared to host the CloudStack Agent and run KVM Instances.

1. Log in to your OS as root.
2. Check for a fully qualified hostname.

   ```
   $ hostname --fqdn
   ```

   This should return a fully qualified hostname such as “kvm1.lab.example.org”. If it does not, edit /etc/hosts so that it does.

3. Make sure that the machine can reach the Internet.

   ```
   $ ping www.cloudstack.org
   ```

4. Turn on NTP for time synchronization.

   Note

   NTP is required to synchronize the clocks of the servers in your cloud. Unsynchronized clocks can cause unexpected problems.

5. Install NTP

   In RHEL or CentOS:

   ```
   $ yum install chrony
   ```

   In Ubuntu:

   ```
   $ apt install chrony
   ```

   In SUSE:

   ```
   $ zypper install chrony
   ```

6. Repeat all of these steps on every hypervisor host.

> **Warning**
>
> CloudStack 4.20 requires Java 17 JRE. Installing CloudStack agent will automatically install Java 17, but it's good to explicitly confirm that the Java 17 is the selected/active one (in case you had a previous Java version already installed) with `alternatives --config java`, after CloudStack agent is installed.

Configure package repository

CloudStack is only distributed from source from the official mirrors. However, members of the CloudStack community may build convenience binaries so that users can install Apache CloudStack without needing to build from source.

If you didn't follow the steps to build your own packages from source in the sections for "Building RPMs from Source" or "Building DEB packages" you may find pre-built DEB and RPM packages for your convenience linked from the downloads page.

Note

These repositories contain both the Management Server and KVM Hypervisor packages.

RPM package repository

There is a RPM package repository for CloudStack so you can easily install on RHEL and SUSE based platforms.

If you're using an RPM-based system, you'll want to add the Yum repository so that you can install CloudStack with Yum.

In RHEL or CentOS:

Yum repository information is found under `/etc/yum.repos.d`. You'll see several `.repo` files in this directory, each one denoting a specific repository.

To add the CloudStack repository, create `/etc/yum.repos.d/cloudstack.repo` and insert the following information.

In the case of RHEL being used, you can replace 'centos' by 'rhel' in the value of baseurl

```
[cloudstack]
name=cloudstack
baseurl=http://download.cloudstack.org/centos/$releasever/4.20/
enabled=1
gpgcheck=0
```

Now you should now be able to install CloudStack using Yum.

In SUSE:

Zypper repository information is found under `/etc/zypp/repos.d/`. You'll see several `.repo` files in this directory, each one denoting a specific repository.

To add the CloudStack repository, create `/etc/zypp/repos.d/cloudstack.repo` and insert the following information.

```
[cloudstack]
name=cloudstack
baseurl=http://download.cloudstack.org/suse/4.20/
enabled=1
gpgcheck=0
```

Now you should now be able to install CloudStack using zypper.

DEB package repository

You can add a DEB package repository to your apt sources with the following commands. Replace the code name with your Ubuntu LTS version : Ubuntu 20.04 (focal), Ubuntu 22.04 (jammy), Ubuntu 24.04 (noble).

Use your preferred editor and open (or create) `/etc/apt/sources.list.d/cloudstack.list`. Add the community provided repository to the file (replace “trusty” with “xenial” or “bionic” if it is the case):

```
deb https://download.cloudstack.org/ubuntu focal 4.20
```

We now have to add the public key to the trusted keys.

```
wget -O - https://download.cloudstack.org/release.asc |sudo tee
/etc/apt/trusted.gpg.d/cloudstack.asc
```

Now update your local apt cache.

```
sudo apt update
```

Your DEB package repository should now be configured and ready for use.

Install and configure the Agent

To manage KVM Instances on the host CloudStack uses a Agent. This Agent communicates with the Management server and controls all the Instances on the host.

> **Note**
>
> Depending on your distribution you might need to add the corresponding package repository for CloudStack.

First we start by installing the agent:

In RHEL or CentOS:

```
$ yum install -y epel-release
$ yum install cloudstack-agent
```

In Ubuntu:

```
$ apt install cloudstack-agent
```

In SUSE:

```
$ zypper install cloudstack-agent
```

The host is now ready to be added to a cluster. This is covered in a later section, see Adding a Host. It is recommended that you continue to read the documentation before adding the host!

If you're using a non-root user to add the KVM host, please add the user to sudoers file:

```
cloudstack ALL=NOPASSWD: /usr/bin/cloudstack-setup-agent
Defaults:cloudstack !requiretty
```

Configure CPU model for KVM guest (Optional)

In additional,the CloudStack Agent allows host administrator to control the guest CPU model which is exposed to KVM Instances. By default, the CPU model of KVM Instance is likely QEMU Virtual CPU version x.x.x with least CPU features exposed. There are a couple of reasons to specify the CPU model:

- To maximise performance of Instances by exposing new host CPU features to the KVM Instances;
- To ensure a consistent default CPU across all machines,removing reliance of variable QEMU defaults;

For the most part it will be sufficient for the host administrator to specify the guest CPU config in the per-host configuration file (/etc/cloudstack/agent/agent.properties). This will be achieved by introducing following configuration parameters:

```
guest.cpu.mode=custom|host-model|host-passthrough
guest.cpu.model=from /usr/share/libvirt/cpu_map.xml(only valid when
guest.cpu.mode=custom)
guest.cpu.features=vmx ept aes smx mmx ht (space separated list of cpu
flags to apply)
```

There are three choices to fulfill the cpu model changes:

1. **custom:** you can explicitly specify one of the supported named model in /usr/share/libvirt/cpu_map.xml
2. **host-model:** libvirt will identify the CPU model in /usr/share/libvirt/cpu_map.xml which most closely matches the host, and then request additional CPU flags to complete the match. This should give

close to maximum functionality/performance, which maintaining good reliability/compatibility if the guest is migrated to another host with slightly different host CPUs.

3. **host-passthrough:** libvirt will tell KVM to passthrough the host CPU with no modifications. The difference to host-model, instead of just matching feature flags, every last detail of the host CPU is matched. This gives absolutely best performance, and can be important to some apps which check low level CPU details, but it comes at a cost with respect to migration: the guest can only be migrated to an exactly matching host CPU.

Here are some examples:

- custom

```
guest.cpu.mode=custom
guest.cpu.model=SandyBridge
```

- host-model

```
guest.cpu.mode=host-model
```

- host-passthrough

```
guest.cpu.mode=host-passthrough
guest.cpu.features=vmx
```

Note

host-passthrough may lead to migration failure,if you have this problem, you should use host-model or custom. guest.cpu.features will force cpu features as a required policy so make sure to put only those features that are provided by the host CPU. As your kvm cluster needs to be made up of

homogeneous nodes anyway (see System Requirements), it might make most sense to use guest.cpu.mode=host-model or guest.cpu.mode=host-passthrough.

Install and Configure libvirt

CloudStack uses libvirt for managing Instances. Therefore it is vital that libvirt is configured correctly. Libvirt is a dependency of cloudstack-agent and should already be installed.

> **Note**
>
> Please note that Cloudstack will automatically perform basic configuration of the agent and libvirt when the host is added. This is relevant if you are planning to automate the deployment and configuration of your KVM hosts.

1. To avoid potential security attack to Instances, We need to turn off libvirt to listen on insecure TCP port. CloudStack will automatically set up cloud keystore and certificates when the host is added to cloudstack. We also need to turn off libvirts attempt to use Multicast DNS advertising. Both of these settings are in `/etc/libvirt/libvirtd.conf`

   Set the following parameters:

   ```
   listen_tls = 0
   ```

   ```
   listen_tcp = 0
   ```

   ```
   tls_port = "16514"
   ```

   ```
   tcp_port = "16509"
   ```

```
auth_tcp = "none"
```

```
mdns_adv = 0
```

2. We have to change the parameters as well:

   On RHEL or CentOS or SUSE modify `/etc/sysconfig/libvirtd`:

   Uncomment the following line:

   ```
   #LIBVIRTD_ARGS="--listen"
   ```

   On RHEL 8 / CentOS 8 / SUSE / Ubuntu / Debian, run the following command :

   ```
   systemctl mask libvirtd.socket libvirtd-ro.socket libvirtd-
   admin.socket libvirtd-tls.socket libvirtd-tcp.socket
   ```

   On Ubuntu 20.04 or older, modify `/etc/default/libvirtd`

   Uncomment and change the following line

   ```
   #libvirtd_opts=""
   ```

   so it looks like:

   ```
   libvirtd_opts="-l"
   ```

   On Ubuntu 22.04 or newer version, modify `/etc/default/libvirtd`:

Uncomment the following line:

```
#LIBVIRTD_ARGS="--listen"
```

Configure libvirt to connect to libvirtd and not to per-driver daemons, especially important on newer distros such as EL9 and Ubuntu 24.04. Edit `/etc/libvirt/libvirt.conf` and add the following:

```
remote_mode="legacy"
```

On Ubuntu 24.04 or newer set libvirtd mode to traditional mode (see https://libvirt.org/manpages/libvirtd.html#system-socket-activation):

```
systemctl mask libvirtd.socket libvirtd-ro.socket libvirtd-
admin.socket libvirtd-tls.socket libvirtd-tcp.socket
```

3. Restart libvirt

   In RHEL or CentOS or SUSE or Ubuntu:

   ```
   $ systemctl restart libvirtd
   ```

Configure the Security Policies

CloudStack does various things which can be blocked by security mechanisms like AppArmor and SELinux. These have to be disabled to ensure the Agent has all the required permissions.

1. Configure SELinux (RHEL, CentOS, SUSE)

   1. Check to see whether SELinux is installed on your machine. If not, you can skip this section.

In RHEL or CentOS, SELinux is installed and enabled by default. You can verify this with:

```
$ rpm -qa | grep selinux
```

2. Set the SELINUX variable in `/etc/selinux/config` to “permissive”. This ensures that the permissive setting will be maintained after a system reboot.

   In RHEL or CentOS:

   ```
   $ vi /etc/selinux/config
   ```

   Change the following line

   ```
   SELINUX=enforcing
   ```

   to this

   ```
   SELINUX=permissive
   ```

3. Then set SELinux to permissive starting immediately, without requiring a system reboot.

   ```
   $ setenforce permissive
   ```

Note

In a production environment, selinux should be set to enforcing and the necessary selinux policies are created to allow the services to run.

1. Configure Apparmor (Ubuntu)

    1. Check to see whether AppArmor is installed on your machine. If not, you can skip this section.

       In Ubuntu AppArmor is installed and enabled by default. You can verify this with:

       ```
       $ dpkg --list 'apparmor'
       ```

    2. Disable the AppArmor profiles for libvirt

       ```
       $ ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/apparmor.d/disable/
       ```

       ```
       $ ln -s /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper
       /etc/apparmor.d/disable/
       ```

       ```
       $ apparmor_parser -R /etc/apparmor.d/usr.sbin.libvirtd
       ```

       ```
       $ apparmor_parser -R /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper
       ```

Configuring the Networking

Warning

This is a very important section, please make sure you read this thoroughly.

Note

This section details how to configure bridges using the native implementation in Linux. Please refer to the next section if you intend to use OpenVswitch

CloudStack uses the network bridges in conjunction with KVM to connect the Guest Instances to each other and the outside world. They also are used to connect the System VMs to your infrastructure.

By default these bridges are called *cloudbr0* and *cloudbr1* etc, but this can be changed to be more descriptive.

Note

Ensure that the interfaces names to be used for configuring the bridges match one of the following patterns: **'eth*', 'bond*', 'team*', 'vlan*', 'em*', 'p*p*', 'ens*', 'eno*', 'enp*', 'enx*'**.

Otherwise, the KVM agent will not be able to configure the bridges properly.

Warning

It is essential that you keep the configuration consistent across all of your hypervisors.

There are many ways to configure your networking. Even within the scope of a given network mode. Below are a few simple examples.

Note

Since Ubuntu 20.04 the standard for managing network connections is by using NetPlan YAML files. Please refer to the Ubuntu man pages for further information and set up network connections figuratively.

Network example for Basic Networks

In the Basic networking, all of the guests in a given pod will be on the same VLAN/subnet. It is common to use the native (untagged) VLAN for the private/management network, so in this example we will have two VLANs, one (native) for your private/management network and one for the guest network.

We assume that the hypervisor has one NIC (eth0) with one tagged VLAN trunked from the switch:

1. Native VLAN for Management Network (cloudbr0)
2. VLAN 200 for guest network of the Instances (cloudbr1)

In this the following example we give the Hypervisor the IP-Address 192.168.42.11/24 with the gateway 192.168.42.1

> **Note**
>
> The Hypervisor and Management server don't have to be in the same subnet

Configuring the Network Bridges for Basic Networks

It depends on the distribution you are using how to configure these, below you'll find examples for RHEL/CentOS, SUSE and Ubuntu.

> **Note**
>
> The goal is to have two bridges called 'cloudbr0' and 'cloudbr1' after this section. This should be used as a guideline only. The exact configuration will depend on your network layout.

Configure RHEL or CentOS for Basic Networks

The required packages were installed when libvirt was installed, we can proceed to configuring the network.

First we configure eth0

```
$ vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

Make sure it looks similar to:

```
DEVICE=eth0
HWADDR=00:04:xx:xx:xx:xx
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
TYPE=Ethernet
BRIDGE=cloudbr0
```

We now have to configure the VLAN interfaces:

```
$ vi /etc/sysconfig/network-scripts/ifcfg-eth0.200
```

```
DEVICE=eth0.200
HWADDR=00:04:xx:xx:xx:xx
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
TYPE=Ethernet
VLAN=yes
BRIDGE=cloudbr1
```

Now that we have the VLAN interfaces configured we can add the bridges on top of them.

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr0
```

Now we configure cloudbr0 and include the Management IP of the hypervisor.

Note

The management IP of the hypervisor doesn't have to be in same subnet/VLAN as the management network, but its quite common.

```
DEVICE=cloudbr0
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=none
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
IPADDR=192.168.42.11
GATEWAY=192.168.42.1
NETMASK=255.255.255.0
STP=yes
```

We configure cloudbr1 as a plain bridge without an IP address

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr1
```

```
DEVICE=cloudbr1
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=none
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
STP=yes
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

Warning

Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configure SUSE for Basic Networks

The required packages were installed when libvirt was installed, we can proceed to configuring the network.

First we configure eth0

```
$ vi /etc/sysconfig/network/ifcfg-eth0
```

Make sure it looks similar to:

```
NAME=eth0
STARTMODE=auto
BOOTPROTO=none
```

We now have to configure the VLAN interfaces:

```
$ vi /etc/sysconfig/network/ifcfg-eth0.200
```

```
NAME=eth0.200
STARTMODE=auto
BOOTPROTO=none
VLAN_ID=200
ETHERDEVICE=eth0
```

Now that we have the VLAN interfaces configured we can add the bridges on top of them.

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr0
```

Now we configure cloudbr0 and include the Management IP of the hypervisor.

Note

The management IP of the hypervisor doesn't have to be in same subnet/VLAN as the management network, but its quite common.

```
NAME=cloudbr0
STARTMODE=auto
BOOTPROTO=static
BRIDGE=yes
BRIDGE_PORTS=eth0
BRIDGE_STP=on
BRIDGE_FORWARDDELAY=5
IPADDR=192.168.42.11
NETMASK=255.255.255.0
```

Add the gateway in `/etc/sysconfig/network/routes`

```
default 192.168.42.1 - cloudbr0
```

We configure cloudbr1 as a plain bridge without an IP address

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr1
```

```
NAME=cloudbr1
STARTMODE=auto
BOOTPROTO=none
BRIDGE=yes
BRIDGE_PORTS=eth0.200
BRIDGE_STP=on
BRIDGE_FORWARDDELAY=5
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

Warning

> Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configure Ubuntu for Basic Networks

All the required packages were installed when you installed libvirt, so we only have to configure the network.

```
$ vi /etc/network/interfaces
```

Modify the interfaces file to look like this:

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

auto eth0.200
iface eth0 inet manual

# management network
auto cloudbr0
iface cloudbr0 inet static
    bridge_ports eth0
    bridge_fd 0
    bridge_stp off
    bridge_maxwait 1
    address 192.168.42.11
    netmask 255.255.255.240
    gateway 192.168.42.1
    dns-nameservers 8.8.8.8 8.8.4.4
    dns-domain lab.example.org

# guest network
auto cloudbr1
iface cloudbr1 inet manual
    bridge_ports eth0.200
    bridge_fd 0
    bridge_stp off
    bridge_maxwait 1
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

Warning

Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Network Example for Advanced Networks

In the Advanced networking mode, it is most common to have (at least) two physical interfaces per hypervior-host. We will use the interface eth0 linked to the bridge ‘cloudbr0’ using the untagged (native) VLAN for hypervisor management. Additionally we configure the second interface for usage with the bridge ‘cloudbr1’ for public and guest traffic. This time there are no VLANs applied by us - CloudStack will add the VLANs as required during actual use.

We again give the Hypervisor the IP-Address 192.168.42.11/24 with the gateway 192.168.42.1

Note

The Hypervisor and Management server don’t have to be in the same subnet

Configuring the Network Bridges for Advanced Networks

It depends on the distribution you are using how to configure these, below you’ll find examples for RHEL/CentOS, SUSE and Ubuntu.

Note

The goal is to have two bridges called ‘cloudbr0’ and ‘cloudbr1’ after this section. This should be used as a guideline only. The exact configuration will depend on your network layout.

Configure RHEL/CentOS for Advanced Networks

The required packages were installed when libvirt was installed, we can proceed to configuring the network.

First we configure eth0

```
$ vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

Make sure it looks similar to:

```
DEVICE=eth0
HWADDR=00:04:xx:xx:xx:xx
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
TYPE=Ethernet
BRIDGE=cloudbr0
```

We now have to configure the second network-interface for use in guest VLANs:

```
$ vi /etc/sysconfig/network-scripts/ifcfg-eth1
```

```
DEVICE=eth1
HWADDR=00:04:xx:xx:xx:xx
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
TYPE=Ethernet
BRIDGE=cloudbr1
```

Now we have the interfaces configured and can add the bridges on top of them.

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr0
```

Now we configure cloudbr0 and include the Management IP of the hypervisor.

Note

The management IP of the hypervisor doesn't have to be in same subnet/VLAN as the management network, but its quite common.

```
DEVICE=cloudbr0
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=none
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
IPADDR=192.168.42.11
GATEWAY=192.168.42.1
NETMASK=255.255.255.0
STP=yes
```

We configure 'cloudbr1' as a plain bridge without an IP address or dedicated VLAN configuration.

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr1
```

```
DEVICE=cloudbr1
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=none
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
STP=yes
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

Warning

> Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configure SUSE for Advanced Networks

The required packages were installed when libvirt was installed, we can proceed to configuring the network.

First we configure eth0

```
$ vi /etc/sysconfig/network/ifcfg-eth0
```

Make sure it looks similar to:

```
NAME=eth0
STARTMODE=auto
BOOTPROTO=none
```

We now have to configure the VLAN interfaces:

```
$ vi /etc/sysconfig/network/ifcfg-eth1
```

```
NAME=eth1
STARTMODE=auto
BOOTPROTO=none
```

Now we have the VLAN interfaces configured we can add the bridges on top of them.

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr0
```

Now we configure cloudbr0 and include the Management IP of the hypervisor.

> **Note**
>
> The management IP of the hypervisor doesn't have to be in same subnet/VLAN as the management network, but its quite common.

```
NAME=cloudbr0
STARTMODE=auto
BOOTPROTO=static
BRIDGE=yes
BRIDGE_PORTS=eth0
BRIDGE_STP=on
BRIDGE_FORWARDDELAY=5
IPADDR=192.168.42.11
NETMASK=255.255.255.0
```

Add the gateway in `/etc/sysconfig/network/routes`

```
default 192.168.42.1 - cloudbr0
```

We configure cloudbr1 as a plain bridge without an IP address

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr1
```

```
NAME=cloudbr1
STARTMODE=auto
BOOTPROTO=none
BRIDGE=yes
BRIDGE_PORTS=eth1
BRIDGE_STP=on
BRIDGE_FORWARDDELAY=5
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

> **Warning**
>
> Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configure Ubuntu for Advanced Networks

All the required packages were installed when you installed libvirt, so we only have to configure the network.

```
$ vi /etc/network/interfaces
```

Modify the interfaces file to look like this:

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

# The second network interface
auto eth1
iface eth1 inet manual

# management network
auto cloudbr0
iface cloudbr0 inet static
    bridge_ports eth0
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1
    address 192.168.42.11
    netmask 255.255.255.240
    gateway 192.168.42.1
    dns-nameservers 8.8.8.8 8.8.4.4
    dns-domain lab.example.org

# guest network
auto cloudbr1
iface cloudbr1 inet manual
    bridge_ports eth1
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1
```

If you are using *netplan* with Ubuntu, below is a sample configuration.

```
$vi /etc/netplan/01-KVM-config.yaml
```

Modify the *YAML* file to look like this:

```
---
network:
  version: 2
  ethernets:
    eth0: {}
    eth1: {}
  bridges:
    cloudbr0:
      addresses:
        - 192.168.42.11/24
      dhcp4: false
      routes:
        - to: default
          via: 192.168.42.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
        search: []
      interfaces:
        - eth0
      parameters:
        stp: true
    cloudbr1:
      dhcp4: false
      interfaces:
        - eth1
      parameters:
        stp: true
```

To apply the above configuration:

```
netplan apply
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

> **Warning**
>
> Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configure the network using OpenVswitch

> **Warning**
>
> This is a very important section, please make sure you read this thoroughly.

In order to forward traffic to your Instances you will need at least two bridges: *public* and *private*.

By default these bridges are called *cloudbr0* and *cloudbr1*, but you do have to make sure they are available on each hypervisor.

The most important factor is that you keep the configuration consistent on all your hypervisors.

Preparing

To make sure that the native bridge module will not interfere with openvswitch the bridge module should be added to the denylist (likely named 'denylist') see the modprobe documentation for your distribution on where to find the denylist. Make sure the module is not loaded either by rebooting or executing rmmod bridge before executing next steps.

The network configurations below depend on the ifup-ovs and ifdown-ovs scripts which are part of the openvswitch installation. They should be installed in /etc/sysconfig/network-scripts/

OpenVswitch Network example

There are many ways to configure your network. In the Basic networking mode you should have two VLANs, one for your private network and one for the public network.

We assume that the hypervisor has one NIC (eth0) with three tagged VLANs:

1. VLAN 100 for management of the hypervisor
2. VLAN 200 for public network of the Instances (cloudbr0)

3. VLAN 300 for private network of the instances (cloudbr1)

On VLAN 100 we give the Hypervisor the IP-Address 192.168.42.11/24 with the gateway 192.168.42.1

> **Note**
>
> The Hypervisor and Management server don't have to be in the same subnet

Configuring the network bridges for OpenVswitch

It depends on the distribution you are using how to configure these, below you'll find examples for RHEL/CentOS.

> **Note**
>
> The goal is to have three bridges called 'mgmt0', 'cloudbr0' and 'cloudbr1' after this section. This should be used as a guideline only. The exact configuration will depend on your network layout.

Configure OpenVswitch

The network interfaces using OpenVswitch are created using the ovs-vsctl command. This command will configure the interfaces and persist them to the OpenVswitch database.

First we create a main bridge connected to the eth0 interface. Next we create three fake bridges, each connected to a specific vlan tag.

```
# ovs-vsctl add-br cloudbr
# ovs-vsctl add-port cloudbr eth0
# ovs-vsctl set port cloudbr trunks=100,200,300
# ovs-vsctl add-br mgmt0 cloudbr 100
# ovs-vsctl add-br cloudbr0 cloudbr 200
# ovs-vsctl add-br cloudbr1 cloudbr 300
```

Configure OpenVswitch in RHEL or CentOS

The required packages were installed when openvswitch and libvirt were installed, we can proceed to configuring the network.

First we configure eth0

```
$ vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

Make sure it looks similar to:

```
DEVICE=eth0
HWADDR=00:04:xx:xx:xx:xx
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
TYPE=Ethernet
```

We have to configure the base bridge with the trunk.

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr
```

```
DEVICE=cloudbr
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
DEVICETYPE=ovs
TYPE=OVSBridge
```

We now have to configure the three VLAN bridges:

```
$ vi /etc/sysconfig/network-scripts/ifcfg-mgmt0
```

```
DEVICE=mgmt0
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=static
DEVICETYPE=ovs
TYPE=OVSBridge
IPADDR=192.168.42.11
GATEWAY=192.168.42.1
NETMASK=255.255.255.0
```

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr0
```

```
DEVICE=cloudbr0
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
DEVICETYPE=ovs
TYPE=OVSBridge
```

```
$ vi /etc/sysconfig/network-scripts/ifcfg-cloudbr1
```

```
DEVICE=cloudbr1
ONBOOT=yes
HOTPLUG=no
BOOTPROTO=none
TYPE=OVSBridge
DEVICETYPE=ovs
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

> **Warning**
>
> Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configure OpenVswitch in SUSE

The required packages were installed when openvswitch and libvirt were installed, we can proceed to configuring the network.

First we configure eth0

```
$ vi /etc/sysconfig/network/ifcfg-eth0
```

Make sure it looks similar to:

```
NAME=eth0
STARTMODE=auto
BOOTPROTO=none
```

We have to configure the base bridge with the trunk.

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr
```

```
NAME=cloudbr
STARTMODE=auto
BOOTPROTO=none
OVS_BRIDGE=yes
```

We now have to configure the three VLAN bridges:

```
$ vi /etc/sysconfig/network/mgmt0
```

```
NAME=mgmt0
STARTMODE=auto
BOOTPROTO=static
OVS_BRIDGE=yes
IPADDR=192.168.42.11
NETMASK=255.255.255.0
```

Add the gateway in `/etc/sysconfig/network/routes`

```
default 192.168.42.1 - mgmt0
```

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr0
```

```
NAME=cloudbr0
STARTMODE=auto
BOOTPROTO=none
OVS_BRIDGE=yes
```

```
$ vi /etc/sysconfig/network/ifcfg-cloudbr1
```

```
NAME=cloudbr1
STARTMODE=auto
BOOTPROTO=none
OVS_BRIDGE=yes
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

Warning

> Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning!

Configuring the firewall

The hypervisor needs to be able to communicate with other hypervisors and the management server needs to be able to reach the hypervisor.

In order to do so we have to open the following TCP ports (if you are using a firewall):

1. 22 (SSH)
2. 1798
3. 16514 (libvirt)
4. 5900 - 6100 (VNC consoles)
5. 49152 - 49216 (libvirt live migration)

It depends on the firewall you are using how to open these ports. Below you'll find examples how to open these ports in RHEL/CentOS and Ubuntu.

Open ports in RHEL / CentOS / SUSE

RHEL and CentOS use iptables for firewalling the system, you can open extra ports by executing the following iptable commands:

```
$ iptables –I INPUT –p tcp –m tcp ––dport 22 –j ACCEPT
```

```
$ iptables –I INPUT –p tcp –m tcp ––dport 1798 –j ACCEPT
```

```
$ iptables –I INPUT –p tcp –m tcp ––dport 16514 –j ACCEPT
```

```
$ iptables -I INPUT -p tcp -m tcp --dport 5900:6100 -j ACCEPT
```

```
$ iptables -I INPUT -p tcp -m tcp --dport 49152:49216 -j ACCEPT
```

These iptable settings are not persistent across reboots, we have to save them first.

```
$ iptables-save > /etc/sysconfig/iptables
```

Warning

latest

On RHEL 8 / CentOS 8 / SUSE, firewalld is the default firewall manager and controls iptables. It is recommended that it be disabled `systemctl stop firewalld ; systemctl disable firewalld`

Warning

On SUSE, iptables are not persisted on reboot, so it is recommended that an iptables and ip6tables service be created to ensure that they persist

Open ports in Ubuntu

The default firewall under Ubuntu is UFW (Uncomplicated FireWall), which is a Python wrapper around iptables.

To open the required ports, execute the following commands:

```
$ ufw allow proto tcp from any to any port 22
```

```
$ ufw allow proto tcp from any to any port 1798
```