Scyld Cloud Manager Documentation

Release 3.1

Penguin Computing

Oct 15, 2019

CONTENTS

CHAPTER

ONE

ADMINISTRATOR'S GUIDE

1.1 Introduction

This guide covers the design, architecture, and administration of Scyld Cloud Manager (SCM). SCM is a robust web-based cloud-management tool composed of multiple inter-dependent proprietary applications, as well as the free and open source software (FOSS) projects Apache, OpenStack, and Ceph.

SCM provides:

- an environment to manage virtual machines, with optional management of users, groups, and storage.
- an optional hosting base for Scyld ClusterWare *head nodes* and peripheral management virtual machine (VM) nodes like job schedulers; when so enabled, it ties together SCM, Scyld ClusterWare and job management into a comprehensive high-performance computing (HPC) cloud solution.
- a robust reporting engine for virtual machine usage, storage, and SCM scheduler job information.

This document is for SCM administrators. It covers:

- SCM architecture and design
- Configuration
- Operation
- Troubleshooting

The following formatting conventions are used:

1. Lines beginning with a `#` and followed by a fixed-width font are shell command lines to be run by the `root` user, or a user with `sudo` permissions. The `#` is an example prompt, and may be different on your system. The rest is the command to be typed. Example:

   # nova list

2. *Italic* items are either terms being defined (such as *GPU*), or symbolic names to be replaced by values (such as *timeout*).
3. `Literal` values are commands or programs (such as `nova` or `Apache`), or actual values (such as using `15` for *timeout* above).

1.2 SCM Architecture

SCM has four major components:

- Scyld Cloud Portal

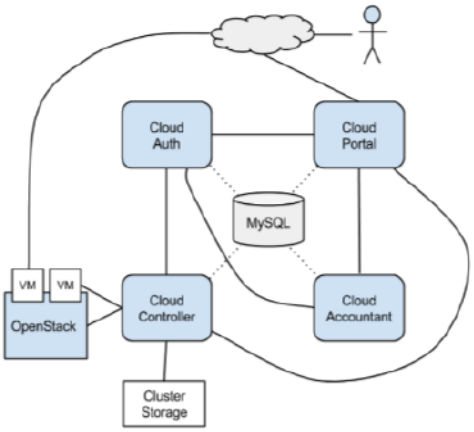
- Scyld Cloud Controller
- Scyld Cloud Auth
- Scyld Cloud Accountant

Each of these is a web application written in Python using the Pyramid web framework, and served by Apache with mod_wsgi. The Cloud Controller, Cloud Auth, and Cloud Accountant all present an application programming interface (API) to be consumed by the Cloud Portal or by individual clients. The Cloud Portal presents a secure web-based interface (HTTPS) for both users and administrators.

1.2.1 Logical Architecture

The following diagram shows the logical architecture of SCM. Each Scyld Cloud application requires access to a MySQL database. The Cloud Portal is the "front end" to the other Scyld Cloud applications, and users typically interact with both the Cloud Portal and the VMs to which they have access.

The Cloud Controller interfaces with OpenStack, storage clusters, and virtual machines. It authenticates all API requests with Cloud Auth.



1.2.2 Physical Architecture

The Openstack services, Ceph, and Galera run on a cluster of 3 controller nodes and a variable number of hypervisors, some with GPUs, comprise the SCM landscape. The SCM web services are generally deployed in a VM. In high-availability (HA) environments, a second SCM VM can be deployed, managed with `HAProxy` and `keepalived` which are used to manage a virtual internet protocol (VIP) address.

[DIAGRAM TBD]

1.3 Administrator's View

As an admin, you'll have more than one entry point into the system:

On the controller node (or nodes, if using dual controllers):

- a web interface to the Scyld Cloud Portal, for SCM orchestration, VM management and accounting information
- an ssh interface to the underlying host operating system and components:
  - OpenStack CLI commands for images, flavors, and VMs

  - SCM, Linux, and OpenStack troubleshooting

On the vmhost nodes:

- an ssh interface to your VMs, for access to your applications.

You won't normally interact directly with Scyld Cloud Controller, Accountant, and Auth. Like the MySQL database and OpenStack services, they are *back end* components that communicate with the front ends and one another. See the Troubleshooting section of this guide for tips on how and when to look into these hidden components if problems arise.

1.4 Components Overview

This is a brief overview of the four main components of Scyld Cloud Workstation. Each has its own detailed section later in this guide.

1.4.1 Scyld Cloud Portal

The Scyld Cloud Portal presents a web-based interface available to users and administrators. The Cloud Portal is a consumer of the public APIs for the other Scyld Cloud components. This means that all of the functionality of the Cloud Portal can also be accomplished by developers consuming the APIs directly.

The Cloud Portal has two different modes:

- The regular user mode is available to all users except the superuser. Using this mode, registered users can create and manage their VMs, storage, secure shell (SSH) keys, and run reports on themselves and any managed users they have.
- The superuser can also log into the Cloud Portal and then has full visibility into SCM: all users and their VMs, with administrative authority over their VMs. The superuser account is always a local account. It is not tied to LDAP (Lightweight Directory Access Protocol) or Active Directory (AD), even if that integration is enabled. The initial superuser login name is admin, and the password is Penguin. For security purposes, you will want to change these defaults at some point; see the Changing User Passwords section later in this Guide.

1.4.2 Scyld Cloud Auth

This is the authorization and authentication component of SCM. It uses the OAuth protocol to issue tokens for authenticated users, and presents those tokens to other SCM components as needed for API access. Users not using the SCM APIs directly may never need to interact with Cloud Auth or understand the use of tokens.

With each API request to an SCM component, a valid token is required. The SCM component will validate the token with Cloud Auth, and also confirm the user's authorization to make the specified API call. Cloud Auth verifies this using Role Based Access Control (RBAC), whereby users are assigned one or more roles that include one or more specific permissions.

1.4.3 Scyld Cloud Controller

The Scyld Cloud Controller handles the actual creation and manipulation of VMs. It does this by interfacing with OpenStack, and by using Ansible to modify packages and configuration files within VMs.

The Scyld Cloud Controller has optional capabilities that can be configured, including:

- Modify and create local users and groups to be shared to a wider audience via network information service (NIS). This capability is not used when LDAP/AD integration is enabled.

- Create and manage user storage volumes that are global to an HPC cluster, and used as user `$HOME` directories. This typically takes the form of mounting and exporting NFS shares, or integrating with a cluster file system like Panasas, Lustre, or Ceph.
- Managing SSH keys in user $HOME storage volumes.

1.4.4 Scyld Cloud Accountant

The Scyld Cloud Accountant serves as the bookkeeper for SCM. It collects, summarizes, and makes available detailed usage metrics for all of its configured resources. For virtual machines, it collects server-hours, or how long a VM has been powered on and available for use. When the Cloud Controller is configured to manage user `$HOME` directories, the metric collected is the maximum daily allocated storage amount.

When HPC cluster integration is present, the Cloud Accountant collects metrics from the HPC job scheduler. Currently, PBS, Sun Grid Engine (SGE), Univa Grid Engine (UGE), and Slurm Workload Manager are supported. Using these metrics, the Cloud Accountant can present summarized data on core-hours: the amount of wall-clock time cluster jobs have used, expressed in terms of equivalent hours on one core. For example, a job taking one hour on one core would be one core-hour, but a job taking one hour and using 12 cores would be 12 core-hours.

1.5 Networking Overview

SCM defines and uses three networks:

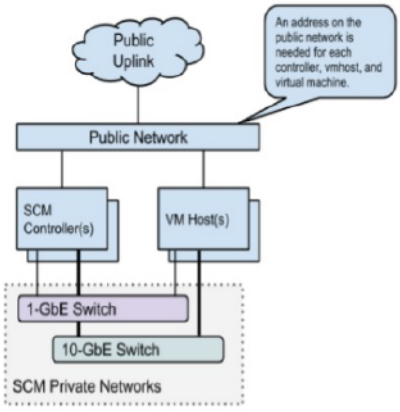
1. The *public network*.
2. A 1-GbE *management* network.
3. A 10-GbE *data* network.

The *public network* is used to allow external access to the Cloud Portal, SCM APIs, and virtual machines. VMs and SCM hosts use the public network to route out to the Internet. Note that the word *public* may not mean that nodes on the public network necessarily have a routable IP address; this may be an internal LAN or Intranet. The public network connections can be 1-GbE or 10-GbE, depending on port availability and uplink connection type.

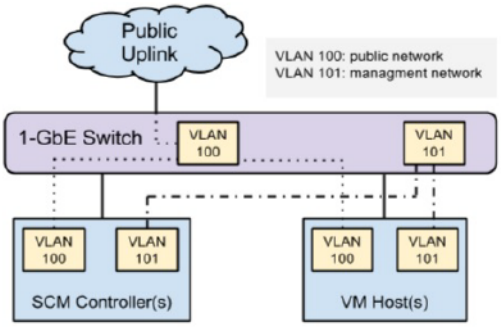
The 1-GbE *management network* is a private network between the SCM nodes. This network is used for SSH and administrative traffic.

The 10-GbE *data network* is used as a high-bandwidth network between the SCM nodes. All inter-node communication between SCM nodes occurs over the data network. This includes Ceph data replication and client storage traffic, OpenStack communication, and MySQL network traffic.

The following diagram shows a logical overview of the SCM networks.



Note that networking can be implemented in many different ways. The public network may be attached via 1-GbE or 10-GbE connections. The public connection may incorporate the use of tagged or untagged virtual local area networks (VLANs). The most flexible method is when using tagged VLANs, and configuring the switches for a "trunked" connection to the SCM nodes. That would enable a configuration like this:



This VLAN example shows that a single 1-GbE switch could be used to handle both the public network and the management network. Additionally, the SCM nodes also support network bonding via link aggregation control protocol (LACP), if multiple connections to a switch are available. This is most useful when two switches are configured as an *active/active* pair via Scyld Cloud Controller (MLAG).

1.6 Virtualization Overview

SCM builds on top of the OpenStack Infrastructure-as-a-Service (IaaS) framework. OpenStack may be configured in many different ways, but SCM configures OpenStack for its needs.

OpenStack is configured to use the kernel-based virtual machine (KVM) hypervisor for virtualization. Each VM has its physical compute properties of boot disk size, number of CPU cores, and amount of random-access memory (RAM) set based on an OpenStack flavor, which are exposed via SCM as server-instance types.

Ceph storage is used to store disk images for OpenStack Glance. These images are raw format disk images that an administrator has uploaded into Glance, and contain a full operating system (OS) installation and potentially additional packages pre-installed. When a new VM is created, its boot disk is created by cloning the disk image from Glance into Cinder, OpenStack's block storage service. Since both Cinder and Glance are configured to use Ceph, this disk cloning can take place using space-efficient copy-on-write (COW) techniques, meaning that new VM disks take up very little space. It is only as files are written and the disk contents diverge from the clone parent, that new storage space is consumed.

Each VM gets its private IP address via dynamic host configuration protocol (DHCP). The IP addresses visible within a VM are private networks only. For external users to access the VMs (i.e. via SSH), an IP address is checked out from a pool of public IP addresses. This "floating IP" address is bound to the node by OpenStack. OpenStack networking then uses NAT to accept incoming traffic for the public IP address and forward it to the VM's private IP address. A set of security group rules filters inbound traffic.

1.7 Scyld Cloud Auth

Scyld Cloud Auth is the component providing a common means for authenticating and authorizing services within an SCM domain. A user must establish an account before making use of any services, generally by registering through the Cloud Portal. If LDAP is configured as the authentication backend, user accounts are created using the configured LDAP attributes. See the LDAP section of this guide for more information.

Users can make use of SCM components via their web-based APIs by first acquiring an authentication token from Scyld Cloud Auth's API. Other SCM components use Scyld Cloud Auth to then authenticate the user by validating

this token and further to determine what level of access the user may have on particular resources.

Scyld Cloud Auth defines roles that encapsulate a set of privileges and assigns users to these roles. It is up to the SCM resource service to match a user request with a role or permission by querying the Scyld Cloud Auth and either grant or deny access.

1.7.1 About OAuth

OAuth is an open standard for authorization designed specifically for web services. It enables a resource owner to authorize third-party access to its resources without directly sharing an end user's credentials. Scyld Cloud Auth makes use of the OAuth protocol solely to authenticate a user; a successful request results in the distribution of a token. Subsequent requests to SCM components use this token, rather than OAuth. The OAuth protocol requires some manipulation of the SCM headers, and clients generally make use of client libraries to handle the complexity.

1.7.2 SCM Roles

The following user roles are supported in SCM:

`superuser` The admin user account is created during installation and assigned to this role. This account has visibility into all users and resources in the portal.

`cloudcontroller_admin` This administrative role is assigned to the user account used by the Cloud Controller.

`cloudaccountant_admin` This administrative role is assigned to the user account used by the Cloud Accountant.

`account_owner` This role permits a regular user to manage additional users on their account. From the Scyld Cloud Portal, an `account_owner` would have visibility into his/her users, managing access to resources, quotas, and additional permissions.

1.7.3 Scyld Cloud Auth Setup

Configuration settings for Scyld Cloud Auth can be found in the `cloudauth.ini` file. Important configuration settings are listed below:

`login_allowed.hosts` Space-separated list of hosts that are permitted to authenticate users via password and allow a user's first login. This should be restricted to the portal only.

`authenticate_allowed.hosts` Space-separated list of hosts that are permitted to authenticate users via password. This should be restricted to SCW hosts. IPs and standard CIDR notation is supported.

`auth_sso` Boolean, default is False. When True, Single Sign-On (SSO) logins through Scyld Cloud Portal are allowed. Additional configuration for Scyld Cloud Portal is required.

1.7.4 LDAP Settings

If SCM is using LDAP for user authentication, then the LDAP settings from the `cloudportal.ini` file need to be in the `cloudauth.ini` file.

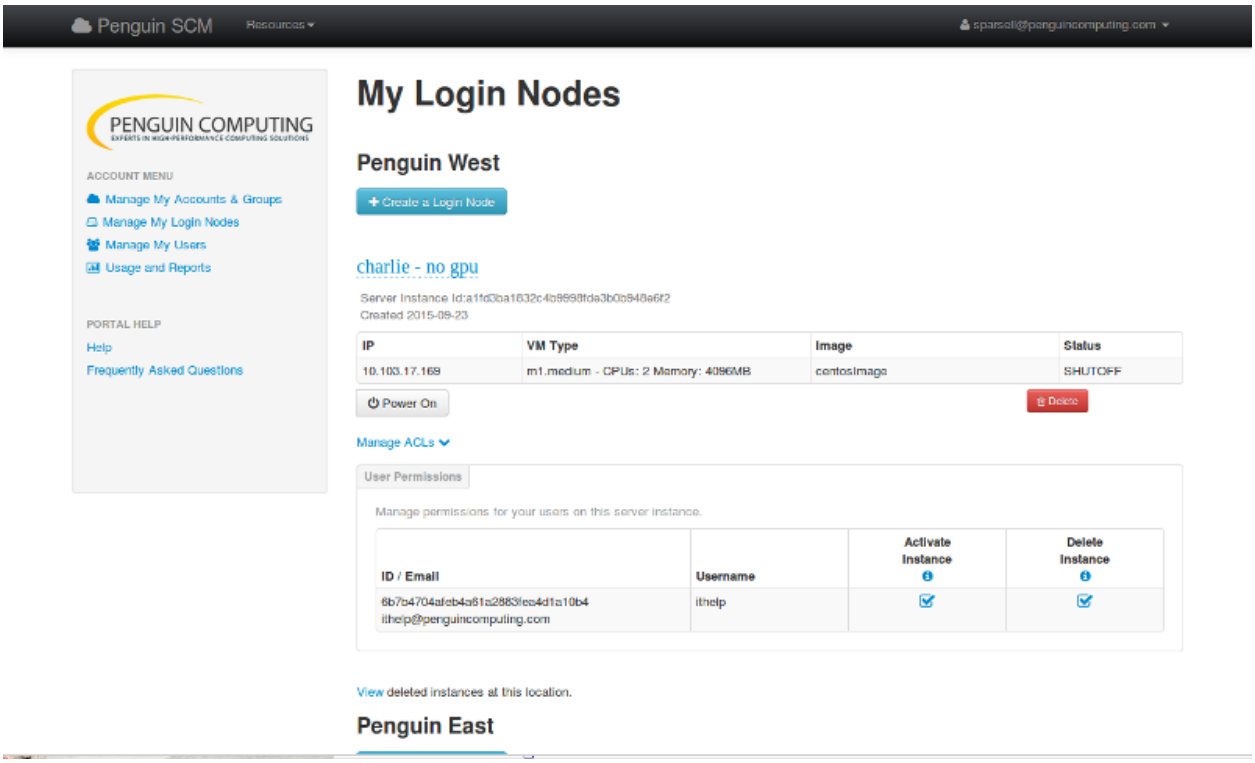
1.7.5 Changing User Passwords

To reset a password for the user account `userid`, including that of the admin user, follow these steps:

```
# source /var/www/wsgi/cloudauth/env/bin/activate
# scmpasswd /var/www/wsgi/cloudauth/scyld-cloud-auth/scyld-cloud-auth/cloudauth.ini␣
↪userid
```

1.8 Scyld Cloud Portal

The SCM portal is a web-based user interface that enables a user to create and manage virtual machines, called server-instances, along with optional management of storage volumes, Unix users and groups, and SSH keys. The SCM portal also enables robust reporting of server-instance utilization and, when enabled, storage and compute cluster core-hour utilization. This is a sample screen image:



All capabilities present within the SCM portal are built upon the SCM APIs, which are documented in the "SCM API Guide."

1.8.1 Scyld Cloud Portal Settings

Configuration settings for the Scyld Cloud Portal can be found in the `cloudportal.ini` file. See the `cloudportal.ini` file for an exhaustive list of settings along with explanations; in most cases the default value is appropriate.

1.8.2 User Registration/Login

New users are created by registering on the portal. There are two workflows defined: LDAP integration and self-registration with email validation. The preferred workflow is indicated in the settings file (`cloudportal.ini`) via the `userauth.driver` setting. Other important registration settings are:

- `registration.type` - Leave blank for unrestricted registration or specify `whitelist` to indicate whitelist-restricted registration
- `default_user_role` - Leave blank or set to `account_owner`. See roles for more information.

1.8.3 LDAP Integration

When using LDAP as an authentication and authorization backend, users bypass registration and simply log in to the portal. If it is their first login, some setup occurs transparently to the user. Their cloudauth account is established and any default roles or SCM resources are configured.

Administrators may use LDAP attributes to limit SCM access by specifying a group DN in the ldap settings (ldap.auth_require_group). Only users belonging to this group may access the portal and SCM resources. If site-wide system (Unix) usernames are maintained in LDAP, the ldap.uname_attr setting should indicate the attribute name.

Important LDAP settings are listed below:

- `ldap.hosts` - Space-separated list of ldap servers
- `ldap.timeout` - Timeout, in seconds, before the next server in the list of ldap.hosts is tried.
- `ldap.cert_file` - Path to TLS certificate.
- `ldap.cacert_file` - Path to TLS certificate chain file.
- `ldap.cacert_dir` - Path to directory of TLS certificates.
- `ldap.bind_uname` - Username of LDAP bind user.
- `ldap.bind_pw` - Password for bind user.
- `ldap.user_search_dn` - Search string for use in resolving a user DN.
- `ldap.uid_attr` - LDAP attribute name for the userid.
- `ldap.mail_attr` - LDAP attribute for the user's email address.
- `ldap.require_group` - Groupname if group membership is required.
- `ldap.require_attr` - LDAP attribute name that must be valued, if listed, for LDAP eligibility.
- `ldap.uname_attr` - LDAP attribute indicating user's system username.

1.8.4 Single Sign-On

When using a SAML Single Sign-On (SSO) for authentication and authorization, users bypass registration and simply log in to the portal. If it is their first login, some setup occurs transparently to the user. Their cloudauth account is established and any default roles or SCM resources are configured.

Any user who is able to reach the portal will be allowed access. The SSO Identity Provider must limit access.

SSO login must also be enabled in Scyld Cloud Auth.

Important SSO settings:

- `auth_sso` - Boolean, defaults to False. Must be True for SSO logins.
- `auth_sso.admin_attributes` - Default is no attributes. Takes a string of newline-separated key = value entries.

`auth_sso.admin_attributes` checks the SAML attributes for a key/value match. If a key has multiple values, each value will be checked. On a match, the user will be able to escalate to the admin local account after logging in by clicking the *Log in as admin* link in the sidebar.. Logging out will return them to their standard profile.

Example:

```
auth.sso.admin_attributes = role = manager@samltest.id
                            role = admin@samltest.id
```

Integration with the Shibboleth Service Provider is discussed in *Shibboleth as Service Provider*.

1.8.5 Self Registration

In this workflow, users register for an account by filling out some basic user information after validating their email address.

1.8.6 Registration Whitelist

The registration.type setting, if set to whitelist will enable the administrator to restrict the ability to register to a specific list of users. This setting may be applied when either registration workflow is used. The "Manage Whitelist" link in the administrative menu will allow the administrator to manage this list.

1.8.7 CloudController Integration

Each SCM installation will have at least one cloudcontroller instance. Each cloudcontroller that the portal will communicate with needs to be listed in the cloudportal.ini file with its particular id.

- `cloudcon.<id>.driver` - Leave this setting "scm". Other values are strictly for testing.
- `cloudcon.<id>.name` - The name displayed in the portal for this cloudcontroller.
- `cloudcon.<id>.description` -The description of this cloudcontroller.
- `cloudcon.<id>.api_version` - Cloudcontroller API version number.
- `cloudcon.<id>.api_endpoint` - hostname:port for the cloudcontroller
- `cloudcon.<id>.api_ssl` - Whether or not the cloudcontroller uses SSL.
- `cloudcon.<id>.report_queues` - A space-separated list of scheduler queues to be visible in the portal, if supported (i.e. cluster integration exists)

1.8.8 Other SCM Options

Other options include the ability for SCM to support integrated storage and support for ssh key management when supporting public key access to VMs. Relevant settings are listed here:

- `scm.ssh_key_mgmt` - Support for ssh key management
- `scm.storage_mgmt` - Support for storage integration.
- `scm.cc_storage_driver` - Type of storage support.

1.8.9 Scyld Cloud Workstation Integration

If the login nodes are running Scyld Cloud Workstation, authentication can be performed through the cloudportal.

- `scw_token_login` - Defaults to False. If True, Scyld Cloud Workstation-enabled login nodes will be accessible through the instances page. Click the SCW Login button for the desired login node.

- `scw_token_login.masq_allowed` - Defaults to False. When False, the SCW Login feature will be unavailable when the admin profile or other superusers are masquerading as another user.

Full documentation of Scyld Cloud Workstation is available at https://updates.penguincomputing.com/scw/scyld-cloud-workstation/

The following settings for SCW on the login node are required:

- `Server.Auth.ScyldCloudAuth.URL` - the Scyld Cloud Auth URL.
- `Server.Auth.ScyldCloudAuth.ApiSecret` - the SCW admin account API key.
- `Server.Auth.ScyldCloudAuth.ApiSecret` - the SCW admin account API secret.

Suggested settings:

- `Server.Auth.ExternalSignInPage` - add the cloudportal URL to disable username/password logins and point logged out users back to the cloudportal.
- `Server.Auth.OSAuthEnabled` - set to False to disable login using login node's operating system authentication.

1.8.10 Portal Branding and Customization

Customizing the look and feel of the SCM portal is possible using the settings in the `cloudportal.ini` file. Details about usage and format can be found in the ini file.

- `site.title` - appears in the top navigation bar and default page title
- `site.logo` - logo file in the main page template
- `site.sidebar_lower` - content (html format) that appears in lower sidebar typically used to display links to external resources, documentation, etc.
- `site.login_help_message` - message appears on login page

Further customization is possible by editing the site templates directly. The site templates use a python library called mako. Care must be taken to avoid syntax errors which may result in broken pages or server errors.

1.9 Scyld Cloud Controller

The Scyld Cloud Controller is directly responsible for the creation and manipulation of VMs through OpenStack. This means defining VMs via Nova, creating their boot volumes in Cinder, and attaching public IP addresses as necessary for external access. The Cloud Controller maintains a mapping of users to VMs, and makes sure that only allowed users are able to access each VM.

The Cloud Controller also has APIs to manage users, groups, and storage volumes, though these are not always used depending on the deployment specifics. Each API resource is described in detail below.

1.9.1 `server-instance`

The `server-instance` resource is for manipulating VMs. In SCM, a server-instance is a VM, and a server-instance directly maps to a VM in OpenStack. Each VM defined in SCM receives a universally unique identifier (UUID) that uniquely identifies the VM.

The UUID used by SCM is not the same as the UUID used by OpenStack. This lets SCM present a persistent VM UUID to a user across multiple power state changes. When a VM is powered off via SCM, the OpenStack VM is actually deleted. All of the VM's configuration information is stored on its boot disk, which is persistent. When the

VM is powered back on via SCM, a brand new OpenStack VM is created, but booted from the pre-existing boot disk. In this way the VM appears persistent, but without SCM creating its own UUID the OpenStack UUID would change every time the VM is powered off.

1.9.2 server-flavor

The `server-flavor` resource retrieves the list of available flavors defined in OpenStack Nova. These flavors are listed in the Cloud Portal as instance-types, and represent the combination of RAM, CPU cores, and boot disk size that the VM will be allocated. This resource is read-only as presented by SCM. New flavors are not defined via the SCM API, but by OpenStack command-line interface (CLI) commands such as `openstack flavor create`, `nova flavor-create`, or through the OpenStack Horizon dashboard.

1.9.3 server-image

The `server-image` resource retrieves a list of available images defined in OpenStack Glance. More accurately, the API returns the list of images that are available for use by the querying user. It is possible to make images in Glance private to a specific user. When this is the case, the superuser sees all images all the time, but a regular user only sees the images that they are allowed access to. This is also a read-only API. New images are defined/uploaded directly using OpenStack CLI commands such as `glance image-create`, `openstack image create`, or through the OpenStack Horizon dashboard.

1.9.4 user

The `user` resource is used to establish a user account with the Cloud Controller. This is necessary in order to create VMs using the Cloud Controller because SCM uses OpenStack's Keystone identity management system. When a user is created in the Cloud Controller, the Cloud Controller creates a user with the same name in Keystone.

Unless a user is using the APIs directly, user creation is a step that is automated by the Cloud Portal. The ultimate result of user creation within the Cloud Controller depends on the back-end user driver configured, which is chosen based on whether LDAP/AD integration is enabled, whether NIS is to be used to make users available to a cluster, or whether local user entries are to be made.

In the case of LDAP/AD integration, creating a user in the Cloud Controller only results in the creation of a user in Keystone. Additionally, the user is associated with the corresponding Cloud Auth user.

When local user creation is enabled, including NIS, creating a user in the Cloud Controller results in a call to `useradd`, such that the user is created in the appropriate `/etc/passwd` file. Using NIS, this entry can then be made globally available to other nodes, such as NFS storage servers, and ClusterWare head nodes and compute nodes.

1.9.5 group

The `group` resource is used to create groups in the local `/etc/group` file. This only makes sense when using the local user and NIS backends, and the group modification APIs are disabled when LDAP/AD integration is enabled.

The Cloud Controller only allows users in the same account owner domain to be in the same group. For example, if Account Owner A has Managed Users B and C, then only users A, B, and C can be added to the group. Account Owner D would not be allowed to be added to the group.

1.9.6 storage-volume

The `storage-volume` resource is designed to be used in conjunction with HPC clusters, such that a created storage-volume maps directly to a user's `$HOME` directory within the cluster. If storage volumes for user `$HOME` directories do not need to be created or managed by SCM, this API can be disabled.

When enabled, having an active storage-volume is a prerequisite for VM creation. The rationale behind this is that a user cannot do any meaningful work without a `$HOME` directory. Storage-volume creation is often the most custom part of an SCM deployment. Many storage environments exist, and the steps needed to create, mount, and export a storage volume can vary greatly. Custom routines can be added to address these steps, and to add in support for things like setting quotas.

1.9.7 user-storage

Rather than individual storage volumes, SCM can manage a storage pool shared between all users of the cluster. In such a setup, the user's storage quota can be set and usage recorded and summarized by the Scyld Cloud Accountant.

The Scyld Cloud Controller server will need to mount the storage pool.

1.9.8 Scyld Cloud Controller Setup

Configuration settings for the Scyld Cloud Controller are in the `cloudcontroller.ini` file. Important configuration settings are:

- `virtualization.boot_timeout` - Time in seconds a VM has to boot into 'ACTIVE' state from a powered off state or creation before it is declared 'ERROR' and configuration is aborted. Defaults to 180.
- `virtualization.reboot_timeout` - Time in seconds a VM has to boot into 'ACTIVE' state from a powered off state or creation before it is declared 'ERROR' and configuration is aborted. Reboots are generally quicker than a first boot. Defaults to 240.
- `virtualization.auto_assign_public_ip` - This boolean flag controls whether SCM will assign a floating IP to the VM once it leaves the 'BOOT' state. Defaults to True.
- `virtualization.flavor_caps` - A dictionary of per-user limits on flavors. Example: {"1": 1} means each user can run 1 server of flavor ID 1.
- `virtualization.hostname_prefix` - The prefix added to the VM's shortened SCC UUID to create the hostname. Defaults to 'vm'.
- `virtualization.hostname_uuid_uniq_chars` - How many characters of the VM's SCC UUID to use when creating the hostname. Defaults to 7.
- `virtualization.image_config_routines` - A map of image IDs to Python classes to perform additional VM configuration after booting.
- `openstack.allow_rdp` - Boolean flag controlling whether to add security group rules allowing RDP (TCP/UDP Port 3389) to the VM. Defaults to False.
- `openstack.autodelete_bootvol` - This boolean flag controls whether the Cinder volume used to boot a VM is automatically deleted when a VM is deleted. Defaults to False.
- `openstack.compute_node_ranges` - Comma seperated list of CIDR ranges to allow interactive qsub traffic from (TCP Port 1024-65535). Example: 10.20.32.0/23,172.16.0.0/16
- `openstack.floating_ip_pool_name` - The name of the public OpenStack network used to create floating IPs.

- `openstack.grow_instances` - Boolean flag controlling whether restarting stopped VMs will grow their root volume to match the VM's flavor's current disk size. Defaults to False.
- `openstack.private_network_name` - The name of the OpenStack network to be used as the primary private network.
- `openstack.secondary_networks` - The name of additional OpenStack networks to be added to the VM as additional interfaces.
- `storage.[mkdir/lustre].options` - The octal file permission for newly created user storage directories. Defaults to '750'.
- `storage.[mkdir/lustre].root` - The base directory where per-user storage directories will be created.
- `storage.[mkdir/lustre].skel` - The location of skeleton files to be copied to all user storage directories.
- `storage.lustre.default_quota` - The default Lustre quota for user storage in MB. Defaults to 0.
- `storage.lustre.max_quota` - The maximum Lustre quota in MB. Defaults to 10e6.
- `storage.lustre.timeout` - The timeout for Lustre operations. Defaults to '3s'.
- `storage.nfs.servers` - For storage drivers that reach out to NFS servers, this is a comma separated list of their IP addresses.
- `storage.nfs.use_exports_d` - Boolean flag. When True, NFS exports are created as seperate files in `/etc/exports.d/`. Otherwise, NFS exports are handled by the NFS storage agent modifying `/etc/exports`. Defaults to True.
- `storage.nfs.export.default_options` - For NFS exports created by the storage driver, this is the default set of export options. Defaults to 'rw,no_subtree_check'.
- `storage.nfs.export.auto_exports` - A dictionary of CIDR:exportoption pairs that serve as a list of exports that should be created for every storage-volume. For example, exporting a newly created $HOME to cluster compute nodes.
- `storage.nfs.export.auto_mounts` - A dictionary of mountpoint:mountoption pairs that should be mounted automatically in every VM created by SCM.
- `storage.nfs.export.auto_clients` - A list of IP addresses that identify nodes that should automatically mount newly created exports. This is typically used for nodes that are *not* login nodes.
- `userauth.manage_sshd_acls` - This boolean flag causes `AllowUsers` lines to be written to the VM's `/etc/ssh/sshd_config` file when the VM starts. If the instance image's property `is_linux` is True (the default), then one line is written for each user account that's permitted to access the instance, and the VM's ssh server is restarted. Defaults to True.
- `userauth.autocreate_storage` - Boolean flag, if True a storage volume will be created during user creation. Defaults to False.

1.9.9 User Storage Usage Recording

If the shared pool user storage driver is enabled, per-user usage needs to be recorded by using the import script after activating the cloud controller's Python environment.

```
# source /var/www/wsgi/cloudcontroller/env/bin/activate
# record_user_storage /var/www/wsgi/cloudcontroller/cloudcontroller.ini
```

Alternatively, if the storage driver is disabled, the cloud controller can still records the user storage usage of a storage pool by passing the pool's mount path directly.

```
# record_user_storage /var/www/wsgi/cloudcontroller/cloudcontroller.ini lustre /mnt/
↪lustre
```

To have this operation performed regularly, this action is usually handled as part of a cron script.

1.10 Scyld Cloud Accountant

An SCM cluster has resources that are consumable by its users, such as allocated storage and core-hours consumed on a compute node. Resource usage is stored in a dedicated database and the billable usage can typically be summarized in daily units over a single metric (such as core-hours, GB, or time). The Scyld Cloud Accountant periodically polls these resource databases, calculates the single usage metric for that particular resource type, and stores it within the accountant database. Users and administrators can then retrieve daily resource usage summaries from the accountant and retrieve results in JavaScript Object Notation (JSON) format using HTTP/HTTPS requests.

1.10.1 Resource Families and Resource Types

Resource types are grouped into resource families if they describe similar resources. Each resource family is expected to have a specific summary metric. Three different resource families are currently supported by the Scyld Cloud Accountant:

- Storage resource usage is defined by the amount of disk bytes allocated.
- User Storage resource usage is defined by the amount of disk bytes used.
- Server-Instance resource usage is defined by the amount of time that a server has been used.
- Job resource usage is defined by the number of core-hours used across all jobs.

There are several types of Job resource types, based on the scheduler:

- PBS Torque
- SGE
- UGE
- Slurm Workload Manager

In addition to daily summary information, the Scyld Cloud Accountant also has the ability to provide more detailed usage information about individual scheduler jobs.

1.10.2 Access Permissions

Access to daily usage summaries is determined by Cloud Auth. While every user is allowed to see their own resource usage information, account owners are also able to query the accountant for resource usage information on their managed users.

1.10.3 Scyld Cloud Accountant Setup

The cloudaccountant must be configured to connect to each cloudcontroller instance in the SCM installation. In most cases, there is a single cloudcontroller. Each cloudcontroller instance has a unique id: the first one is id 1, the next is id 2, etc.

- `sqlalchemy.cloud.controller-id.resource-type.url` - Specifies the database URL for the resource-type for a given cloud-container-id. Currently assumes MySQL databases. For example: To point to the server-instances resource-type of cloud-controller-id 1, one would set sqlalchemy.cloud.1.instances.url = url
- `sqlalchemy.cloud.controller-id.slurm.server` - Specifies a SSH URL capable of running `sacct` for the Slurm job manager

1.10.4 Initializing the Scyld Accountant Database

Note: This only applies to MySQL databases that have not been initialized.

If the Scyld Cloud Accountant database is ever destroyed and needs to be recreated, you can do so by running the database initialization script. Begin by logging in as root and activating the python environment for the accountant:

Verify that `sqlalchemy.accountant.url` has been properly defined in `cloudaccountant.ini`.

Now run the initialization script:

```
# source /var/www/wsgi/cloudaccountant/env/bin/activate
# initialize_scyld_cloud_accountant_db /var/www/wsgi/cloudaccountant/cloudaccountant.
↪ini
```

1.10.5 Importing from Resource Databases

To manually import data from the resource databases into the cloud accountant databases, first, activate the Python environment for the accountant, then run the import script:

```
# source /var/www/wsgi/cloudaccountant/env/bin/activate
# scyld_cloud_accountant_db_import /var/www/wsgi/cloudaccountant/cloudaccountant.ini
↪<START DATE> <END DATE>
```

This will import data relevant to all resources with URLs listed in the cloudaccountant.ini file between the given <START DATE> and <END DATE>.

NOTE: Importing over the same dates repeatedly will not remove any existing entries or produce duplicate rows. Importing will only add or modify existing entries.

To have this operation performed regularly, this action is usually handled as part of a cron script.

1.10.6 Importing from Slurm Accountant

To manually import data for from the resource databases into the cloud accountant databases, first, activate the Python environment for the accountant, then run the import script:

```
# source /var/www/wsgi/cloudaccountant/env/bin/activate
# scyld_cloud_accountant_slurm_import /var/www/wsgi/cloudaccountant/cloudaccountant.
↪ini -r <START DATE>:<END DATE>
```

This will import data from the Slurm servers listed in the cloudaccountant.ini file between the given <START DATE> and <END DATE>.

NOTE: Importing over the same dates repeatedly will not remove any existing entries or produce duplicate rows. Importing will only add or modify existing entries.

To have this operation performed regularly, this action is usually handled as part of a cron script.

1.11 Working with OpenStack

SCM controller nodes have full administrator access to the OpenStack CLI when logged in as the root user. To enable this access, run this command on a controller:

```
# source ~/oscreds.sh
```

You will run OpenStack CLI commands to list, create, or modify images and flavors. SCM will handle the higher-level details of creating and deleting VMs from these base elements.

Full documentation of OpenStack commands is at the official site. The following sections are brief summaries.

1.11.1 Horizon Dashboard

OpenStack also offers a web dashboard, Horizon, for most but not all admin operations. The dashboard will be hosted on the same server that provides the OpenStack authentication URL.

1.11.2 Images

1.11.2.1 Listing Images

This command lists images that the Keystone-authenticated user can access:

```
# glance image-list
+--------------------------------------+---------------+-------------+-----------------
↪--+-------------+--------+
| ID                                   | Name          | Disk Format | Container␣
↪Format | Size        | Status |
+--------------------------------------+---------------+-------------+-----------------
↪--+-------------+--------+
| 26322b71-4e67-45d4-9662-2ba58b55f5dd | apache        | raw         | bare          ␣
↪   | 21474836480 | active |
| 3bb96ae9-6b7f-411d-9822-1f5e171c77b8 | CentOS-Apache | raw         | bare          ␣
↪   | 2147483648  | active |
| e8568161-cf98-4601-ac20-24d875fc995a | centosImage   | raw         | bare          ␣
↪   | 2147483648  | active |
+--------------------------------------+---------------+-------------+-----------------
↪--+-------------+--------+
```

If you are authenticated as an admin, you can see all images by adding the `--all-tenants` argument.

1.11.2.2 Creating an Image

This is the command to make new images for OpenStack/Glance, and later used to create VMs in Scyld Cloud Portal. You provide a base image (single-file version of an operating system) and various parameters to create a Glance Image. This image's data will be stored in our back-end storage system, such as Ceph, and the metadata describing the image will be stored in MySQL. Example:

```
# glance image-create \
    --name my_new_image \
    --is-public True \
    --file /root/images/my_image.raw \
    --disk-format raw \
```

(continues on next page)

(continued from previous page)

```
    --container-format bare \
    --min-disk 15 \
    --min-ram 512 \
    --progress
```

Some of the arguments include:

- `name` – What to call your new image. It doesn't need to be unique, but makes it easier to refer to the image in other commands if it is.
- `is-public` – *True* if you want this image to be visible for use
- `file` – file system location of the source image
- `disk-image` – format of the source image file. `raw` is the output of the `dd` command.
- `container-format` – extra file format type; just use `bare`.
- `min-disk` – minimum required disk space, in gigabytes
- `min-ram` – minimum required memory, in megabytes
- `progress` – show image creation status time bar

1.11.2.3 Showing Image Details

You can print more details about an existing image:

```
# glance image-show image_id
```

image_id can be the long hex string, or the name (if it's unique).

1.11.2.4 Modifying an Image

You can modify an existing Glance image for various purposes: making a test image public, adding a property to describe a special feature of the image, and so on. In this example, make a private test image publicly available:

```
# glance image-update \
    --is-public True \
    my_private_image
```

1.11.3 Flavors

1.11.3.1 Listing Flavors

This command shows all of the currently defined flavors and their standard attributes:

```
# nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_
↪Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True  ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True  ␣
↪   |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True  ␣
↪   |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True  ␣
↪   |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True  ␣
↪   |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
```

Any flavor can have *extra_specs* – one or more *name*:*value* pairs to indicate some feature or restriction of the underlying hardware, such as the presence or absence of a GPU. In this example, we modified the standard flavors with some extra information. See the following section (Modifying an Existing Flavor) to see how to do this.

```
# nova flavor-list --extra-specs
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+-------------------------------------------------------------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_
↪Public | extra_specs                                                 |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+-------------------------------------------------------------+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True  ␣
↪   | {u'gpu': u'False'}                                          |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True  ␣
↪   | {u'gpu': u'False'}                                          |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True  ␣
↪   | {u'gpu': u'False'}                                          |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True  ␣
↪   | {u'gpu': u'True', u'pci_passthrough:alias': u'K2_Grid:1'} |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True  ␣
↪   | {u'gpu': u'False'}                                          |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+-------------------------------------------------------------+
```

1.11.3.2 Creating a Flavor

To create new flavors in OpenStack, use the `nova flavor-create` command. It has this syntax:

```
nova flavor-create \
   --is-public [True|False] \
   name \
   id \
   ram_mb \
   disk_gb \
   vcpus
```

Note: Other than the `is-public` argument, the others are order-dependent.

The arguments are:

- `is-public` – should this flavor be visible to others (True) or not (False)?
- `name` – give this image a memorable name

- `id` – specify a unique integer id; you can use one more than the highest existing flavor id
- `ram_mb` – the maximum amount of RAM, in megabytes
- `disk_gb` – the maximum amount of disk storage, in gigabytes
- `vcpus` – the maximum number of CPUs

Example:

The highest flavor id (from `nova flavor-list` above) was 5, so we'll use 6 for our new id. We'll name the image `scm.login_small`, with 512 MB of RAM, 5 GB of disk, and one CPU. Also, make it public:

```
# nova flavor-create \
     --is-public True \
     scm.login_small \
     6 \
     512 \
     5 \
     1

+----+------------------+-----------+------+-----------+------+-------+-------------+--
↪---------+
| ID | Name             | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |␣
↪Is_Public |
+----+------------------+-----------+------+-----------+------+-------+-------------+--
↪---------+
| 6  | scm.login_small | 512       | 5    | 0         |      | 1     | 1.0         |␣
↪True      |
+----+------------------+-----------+------+-----------+------+-------+-------------+--
↪---------+
```

1.11.3.3 Modifying a Flavor

You can change the standard values or extra_specs of a flavor:

```
# nova flavor-key \
     flavor \
     action \
     key=value
```

- `flavor` – the numeric ID or Name from the
- `action` – `set` to assign a key, `unset` to delete it
- *key* – an extra flavor attribute to assign or delete a *value*

Example:

```
# nova flavor-key 6 set gpu=true
```

1.11.3.4 Public IP Addresses

Each instance can have one or more private IP addresses and one public one. Private IP addresses are used for communication between instances and internal infrastructure, and public ones are used for communication with the outside world. When you launch an instance, it is automatically assigned a private IP address that stays the same until you explicitly terminate the instance. Rebooting an instance has no effect on the private IP address.

A pool of floating IPs, configured by the SCM administrator, is available to instances.

SCM will automatically allocate and assign a public IP address from the pool of floating IPs to each instance as it is created.

If no more floating IP addresses are available in the pool, the VM will show as "FAILED" in its status, and will be unreachable to external users.

To show the list of IP addresses in the pool, find which OpenStack network is used for public IPs. This is the openstack.floating_ip_pool_name setting in the SCM cloud controller. For example, 'Public'.

```
# openstack subnet list --network Public
+--------------------------------------+--------------+--------------------------------
↪-------+----------------+
| ID                                   | Name         | Network                        ␣
↪       | Subnet         |
+--------------------------------------+--------------+--------------------------------
↪-------+----------------+
| dd86cda6-a496-4732-9299-6497aca6aa94 | PublicSubnet | 4010092c-2a8e-46f8-972a-
↪55ffa1138563 | 203.0.113.0/24 |
+--------------------------------------+--------------+--------------------------------
↪-------+----------------+

# openstack subnet show -c allocation_pools dd86cda6-a496-4732-9299-6497aca6aa94
+------------------+---------------------------+
| Field            | Value                     |
+------------------+---------------------------+
| allocation_pools | 203.0.113.12-203.0.113.199 |
+------------------+---------------------------+
```

1.12 Linux Image Preparation for OpenStack

This describes how to prepare a Linux guest VM for SCM. CentOS is used in the examples, but any version of Linux can be used as a guest VM.

1.12.1 Install CentOS manually

- Use the CentOS netinstall ISO and the following URLs for the packages:
  - http://mirror.centos.org/centos/6/os/x86_64/
  - http://mirror.centos.org/centos/7/os/x86_64/
- Create an account. In these examples, we'll use the account name `sam`.
- Choose workstation software group.
- Skip KDE.
- After installation, log in as `sam`.
- Add `sam` to the `wheel` group.
- `visudo` to enable `%wheel` without password.
- As root:
  - `yum update`
  - `yum install epel-release`

1.12.2 Network

Edit the `/etc/sysconfig/network-scripts/ifcfg-eth0` file. Remove:

- UUIDs
- MAC addresses (assigned by OpenStack)

Change:

```
NM_CONTROLLED="no"
```

Verify:

```
ONBOOT="yes"
BOOTPROTO
```

```
# yum remove NetworkManager
```

Finally, remove this file:

```
# rm /lib/udev/rules.d/75-persistent-net-generator.rules
```

This prevents the automatic creation of a `/etc/udev/rules.d/70-persistent-net.rules` file, which would rename network interfaces and interfere with the reuse of SCM images.

1.12.3 fstab

Replace UUIDs in `/etc/fstab` with device names like `/dev/vda` for the `/boot` partition, so the system will boot when replicated to a new boot volume.

1.12.4 nVidia GPU steps (optional)

If you are using an nVidia GPU:

```
# echo "blacklist nouveau" >> /etc/modprobe.d/blacklist.conf
```

Add these lines to `/etc/grub.conf`:

```
serial --unit=0 --speed=115200
terminal --timeout=10 console serial
```

Change each kernel line by replacing `rhgb quiet` at the end of each line with:

```
rdblacklist=nouveau console=tty0 console=ttyS0,115200n8
```

Install driver prerequisites:

```
# yum install dkms
# yum groupinstall "Development Tools"
```

Download the nVidia drivers:

```
# wget http://us.download.nvidia.com/XFree86/Linux-x86_64/352.41/NVIDIA-Linux-x86_64-
↪352.41.run
```

Drop back to runlevel 3:

```
# telinit 3
```

Install the nVidia drivers:

```
# chmod +x NVIDIA-Linux-x86_64-352.41.run
# ./NVIDIA-Linux-x86_64-352.41.run --silent --dkms
```

Edit `/etc/inittab` and change the default runlevel from 5 to 3.

1.12.5 Install Scyld Cloud Workstation (optional)

Obtain the appropriate version of SCW from Penguin Computing as a RPM.

Install the RPM

yum localinstall scyld-cloud-workstation-$VERSION-rpm.

Configure SCW following the User Guide - http://www.penguincomputing.com/documentation/scyld-cloud-workstation/user-guide/

1.12.6 Install cloud-init

Install the prerequisites for cloud-init:

```
# yum install \
     acpid cloud-init cloud-utils cloud-utils-growpart \
     dracut-kernel dracut dracut-modules-growroot
```

Edit `/etc/cloud/cloud.cfg` and change the values at the top of the file to these:

```
disable_root: 0
ssh_pwauth:   1
ssh_deletekeys:   0
```

Empty the `/var/lib/cloud` directory for the full cloud-init run on first boot:

```
# rm -rf /var/lib/cloud/*
```

1.12.7 Integrate Scyld Cloud Workstation with SCM (optional)

Scyld Cloud Workstation can be integrated with SCM so only the user who created the VM can login and/or is automatically logged into their desktop after authenticating with Scyld Cloud Workstation.

SCM stores the username and email of the user who created the VM in OpenStack metadata, available from inside the VM as JSON at http://169.254.169.254/openstack/latest/meta_data.json

Relevant content of example meta_data.json:

```
{"meta": {
  "scm_username": "testuser",
  "scm_users": "testuser@example.com"
  },
}
```

As discussed in the SCW User Guide, if using Scyld CloudAuth for SCW authentication, the **Server.Auth.ScyldCloudAuth.Allow** config setting controls which users can authenticate to the SCW instance. Example relevant lines allowing login by username or email:

```
<config>
 <Server>
  <Auth>
   <ScyldCloudAuth>
    <URL>https:/auth.scm.example.com</URL>
    <Allow>
     <Username>testuser</Username>
     <Username>testuser@example.com</Username>
    </Allow>
   </ScyldCloudAuth>
  </Auth>
 </Server>
</config>
```

Penguin Computing can provide a cloud-init script that reads from the metadata and configures **Server.Auth.ScyldCloudAuth.Allow** setting from the OpenStack metadata.

By default, after authenticating Scyld Cloud Workstation presents the user with a login screen. The login node can be customized so the user will be automatically logged in.

For CentOS 6 using the GNOME desktop with GDM, autologin is configured in `/etc/gdm/custom.conf` as:

```
[daemon]
AutomaticLoginEnable = true
AutomaticLogin = testuser
```

For Centos 7 using MATE and LightDM, autologin can be configured in `/etc/lightdm.conf.d/50-autologin.conf` as:

```
[Seat:*]
autologin-user=testuser
autologin-user-timeout=0
```

Penguin Computing can provide a cloud-init script that reads from the metadata and customizes the appropriate autologin setting from the OpenStack metadata.

1.12.8 Shutdown the VM

```
# shutdown -h now
```

1.12.9 Create a CentOS VM

With virt-manager, create a CentOS VM, using a qcow2 file for the boot disk. Details are available here.

1.12.10 Prep the image for reuse

Get the name of the new VM:

```
# virsh list --all
```

Prep the image:

```
# virt-sysprep -d centos-6.7
```

1.12.11 Install libguestfs

```
# yum install libguestfs-tools
```

1.12.12 Compress the image to upload

```
# qemu-img convert -c \
    -f qcow2 \
    -O qcow2  \
    /var/lib/libvirt/images/centos-6.7-scw.qcow2 \
    /tmp/centos-6.7-scw.qcow2
```

1.12.13 Convert qcow2 to a sparse raw file for Glance

```
# qemu-img convert \
    -f qcow2 \
     -O raw \
     /tmp/centos-6.7-scw.qcow2 \
     /tmp/centos-6.7-working.raw
```

1.12.14 Load the raw file into Glance

```
# glance image-create \
    --name centos_image  \
    --is-public true \
    --file /tmp/centos6.7-working.raw  \
    --disk-format raw \
    --container-format bare \
    --min-disk 8 \
    --min-ram 512 \
    --progress \
    --property image_type=loginnode \
    --property hw_video_model=qxl \
    --property hw_video_ram=8
```

If the image will include Scyld Cloud Workstation, add *–property service_url='https://{}/'*

1.12.15 Modifying image file properties

Once the image is uploaded, it can be updated through the Horizon dashboard, *openstack image set <IMAGE_ID>*, or *glance image-update <IMAGE_ID>*

1.13 Windows Image Preparation for OpenStack

1.13.1 Prerequisites

Windows requires the use of Scyld Cloud Workstation and cloudbase-init.

1.13.2 GPU Support

SCW supports the use of NVIDIA GRID SDK compatibile graphics cards for 3D acceleration.

This requires:

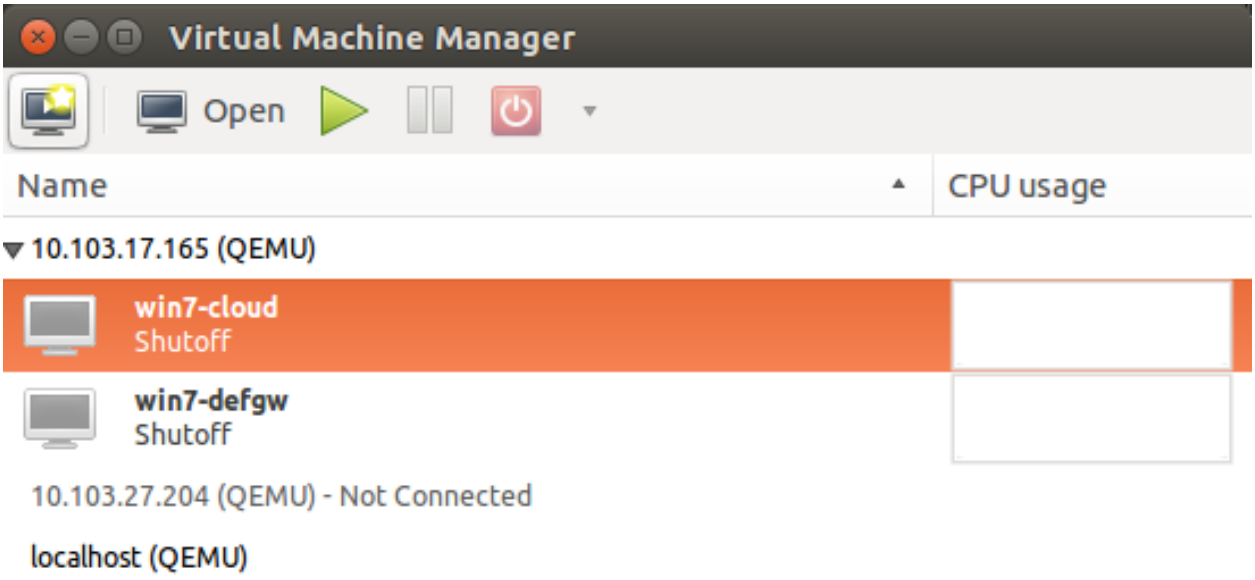
- an Openstack cluster, configured for GPU passthrough
- an OpenStack vmhost with a passthrough-capable graphics card

You will also need a desktop-enabled machine with passwordless ssh access to the individual nodes in the Openstack cluster, and with a working virt-manager installation.

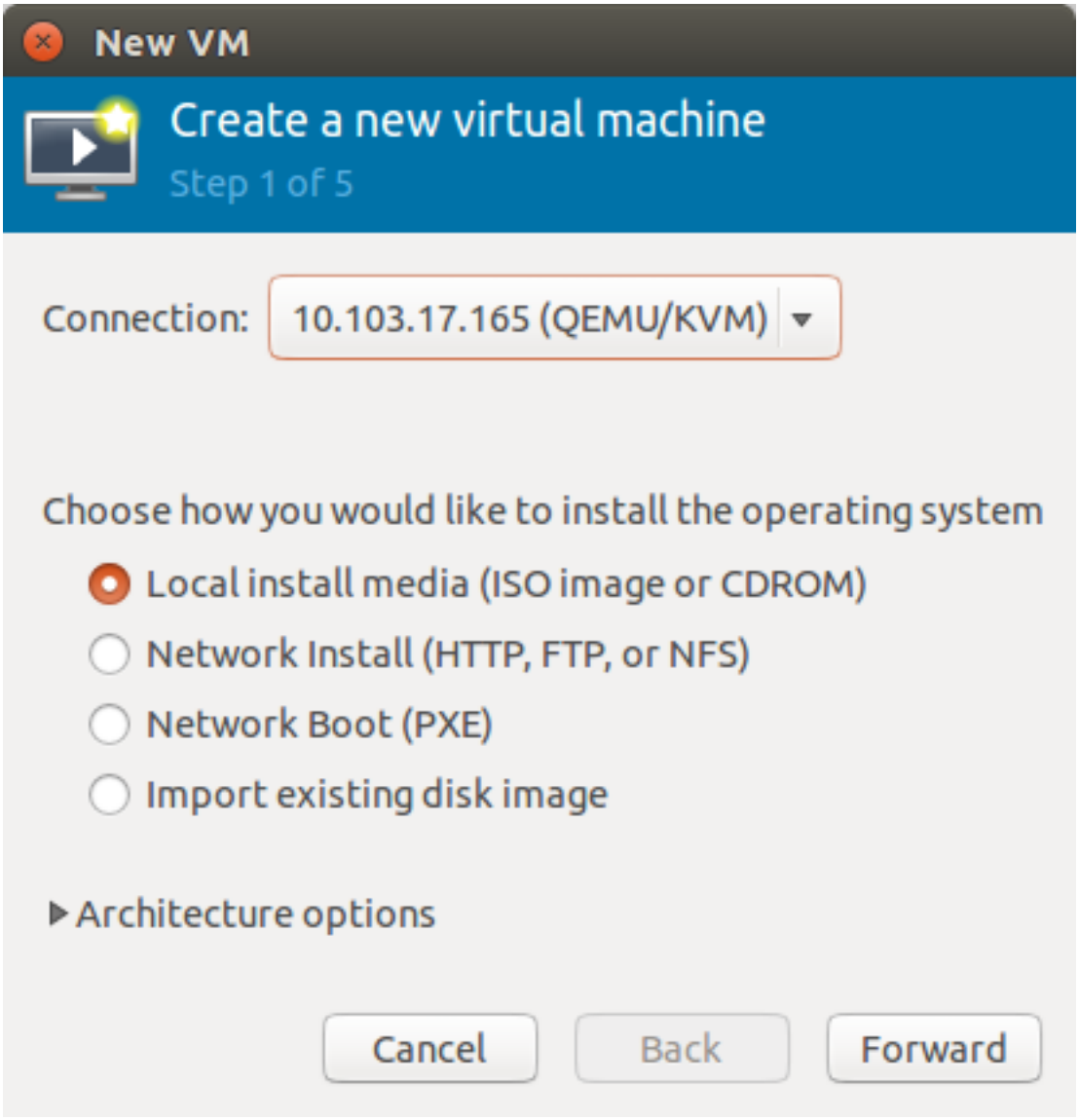
The NVIDIA SDK drivers for Windows should also be accessible by SCP with a password somewhere on the host machine.

1.13.3 VM Setup

In virt-manager, add a connection to the vmhost. if not already present. (File -> Add Connection. Hypervisor QEMU/KVM, method SSH, username root).

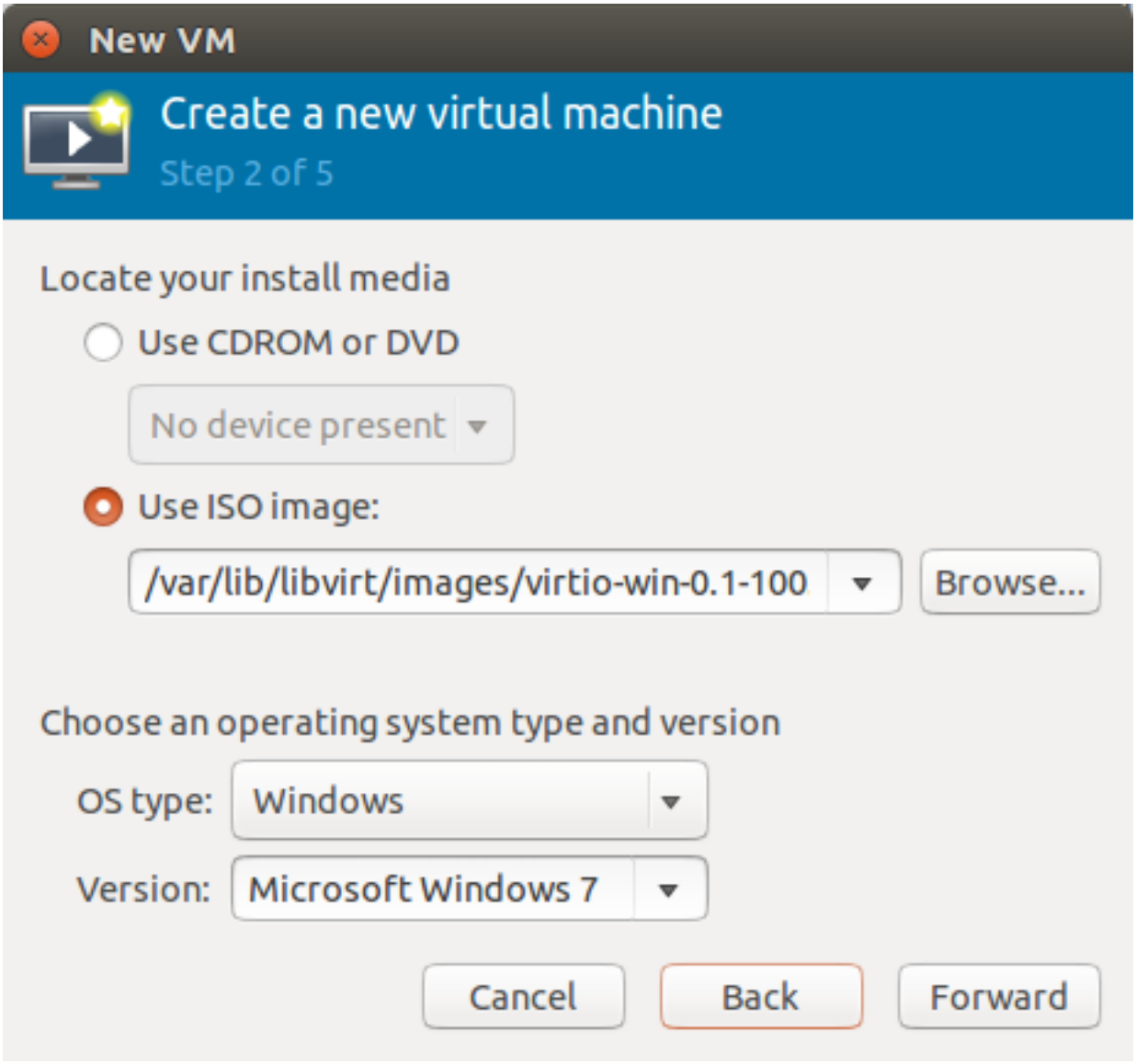


Create a new virtual machine (File -> New Virtual Machine). Select the connection corresponding to the vmhost and the "Local install media" option.

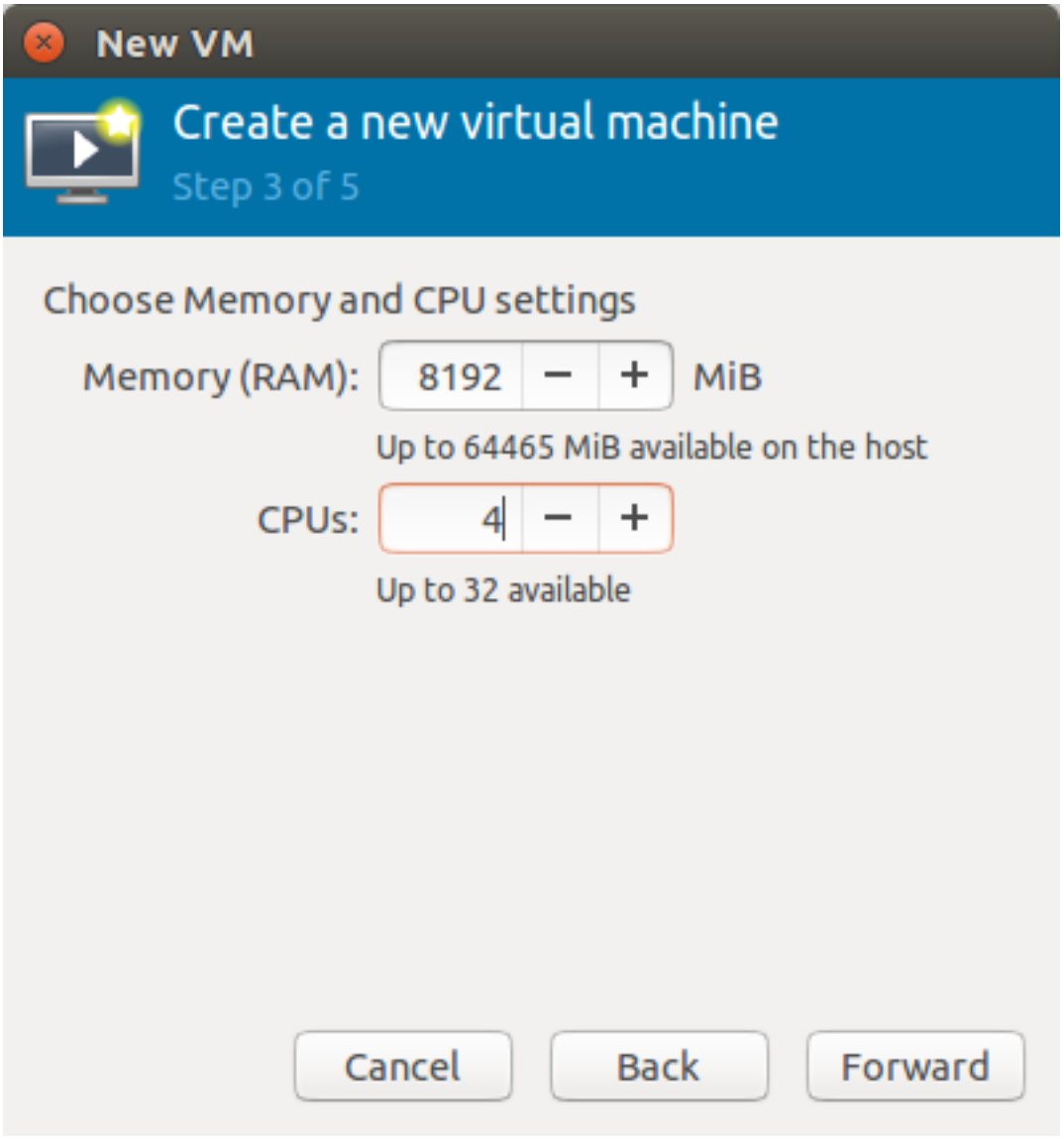


Select the virtio drivers disk. Note that we're not choosing the actual Windows install disk, as we'll be adding it later, and a bug in virt-manager causes it to configure new VMs to boot from whichever disk was last added. If this is fixed in future versions, then you should specify the Windows disk here and the driver disk later.

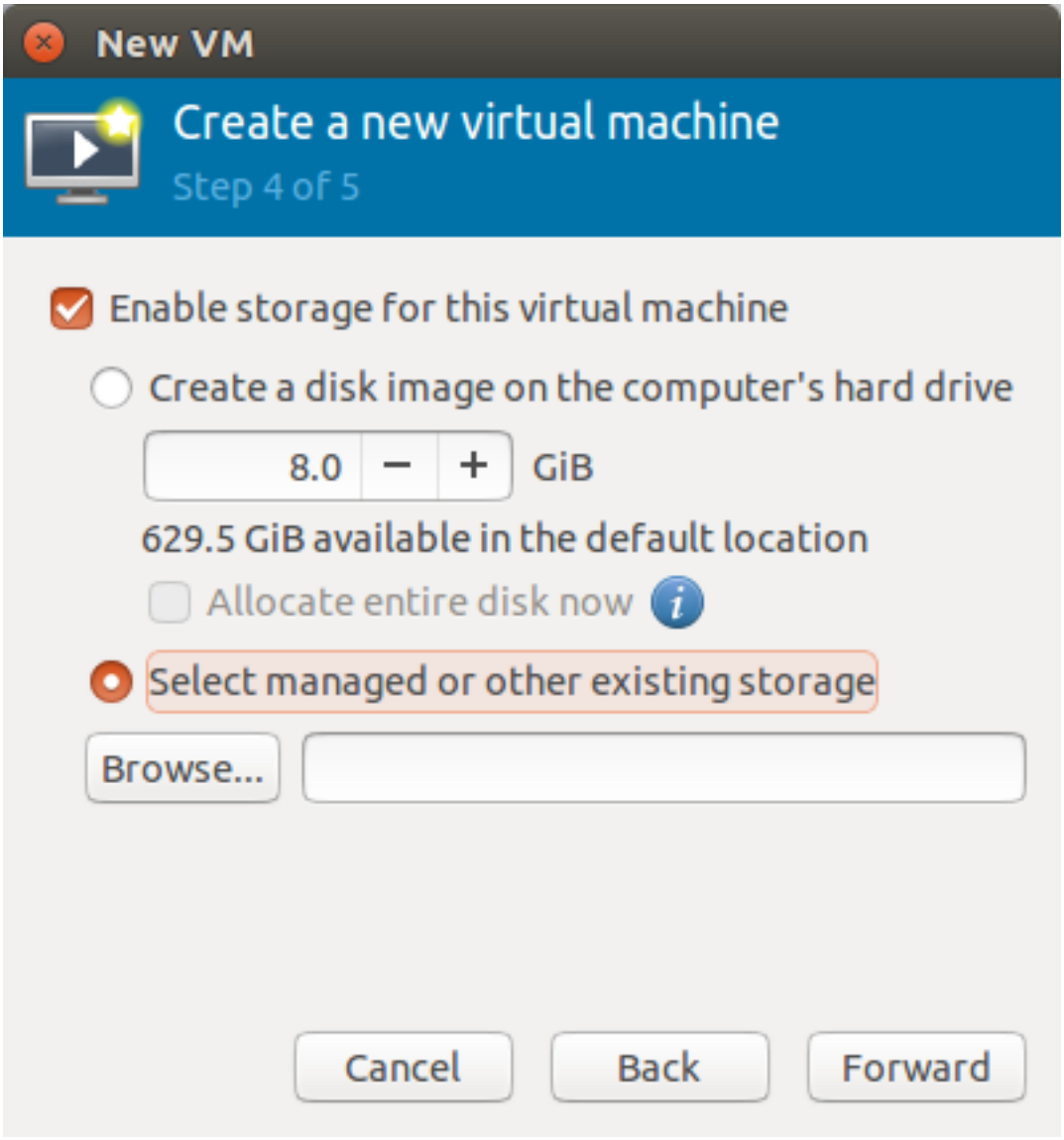
Select the appropriate Windows version.



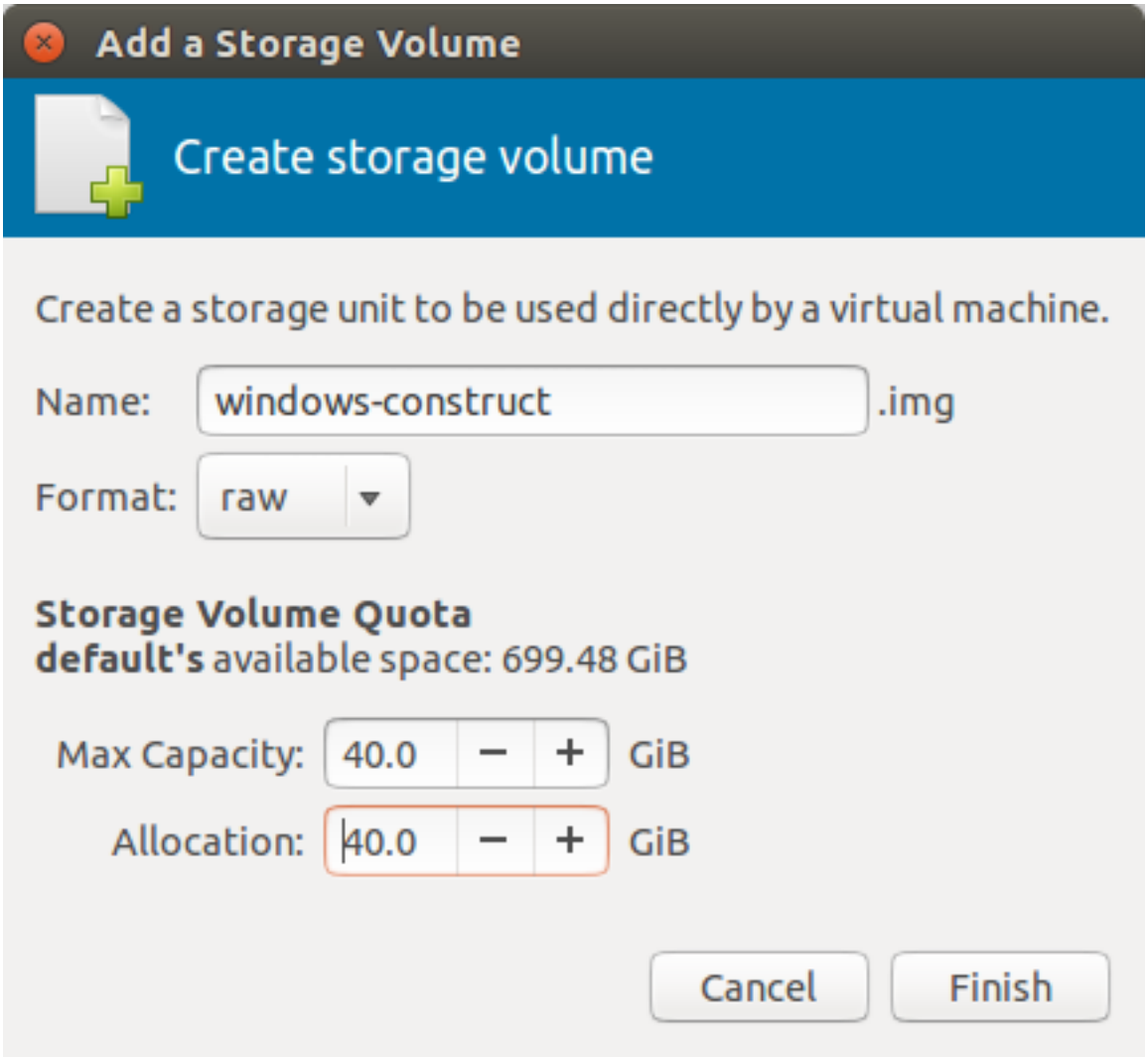
Allocate an appropriate amount of memory and number of cores. This will impact the image we eventually export; we just need enough to immediately install Windows.



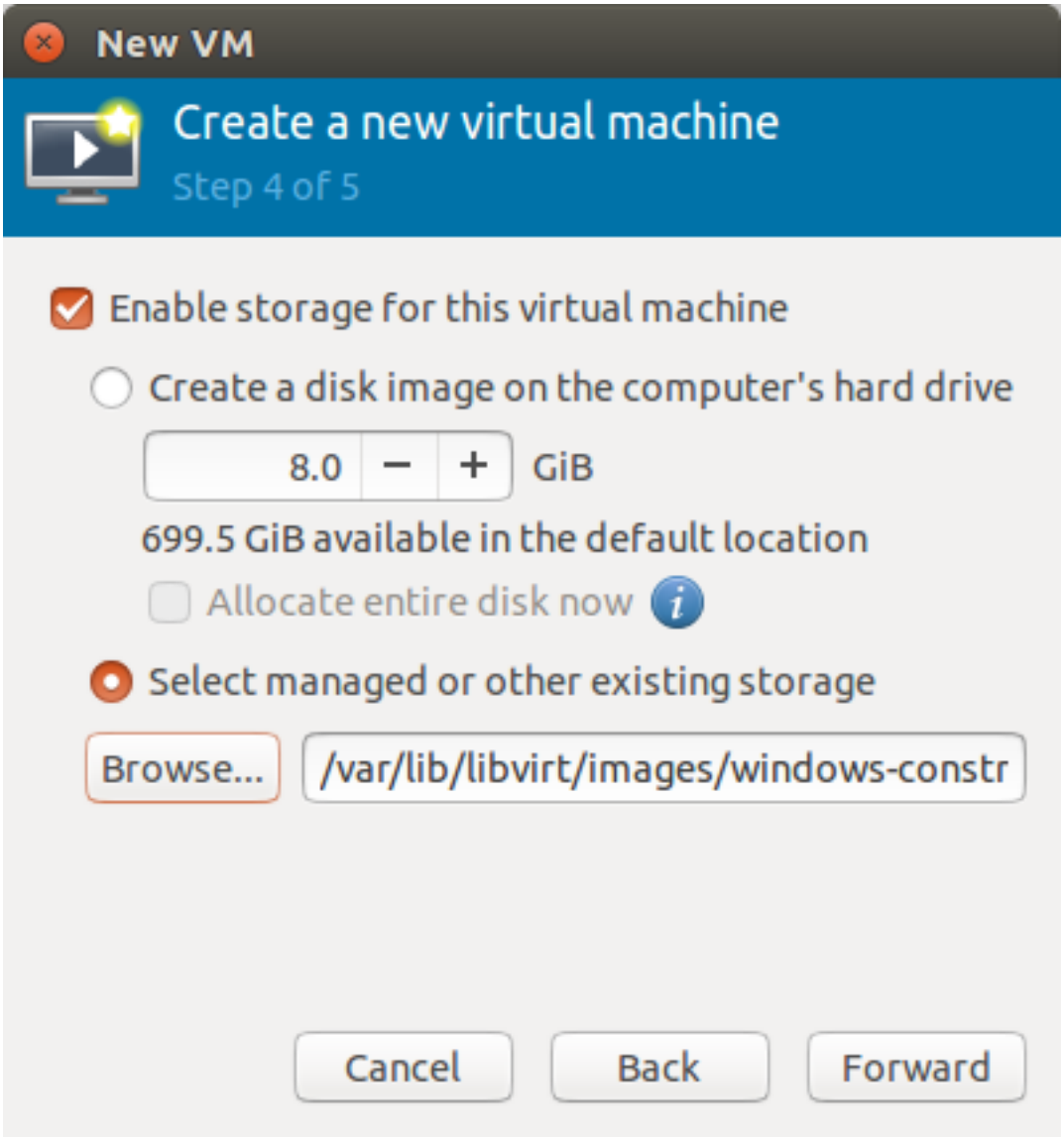
At the next step, select managed or other existing storage. We'll be allocating a new volume manually.



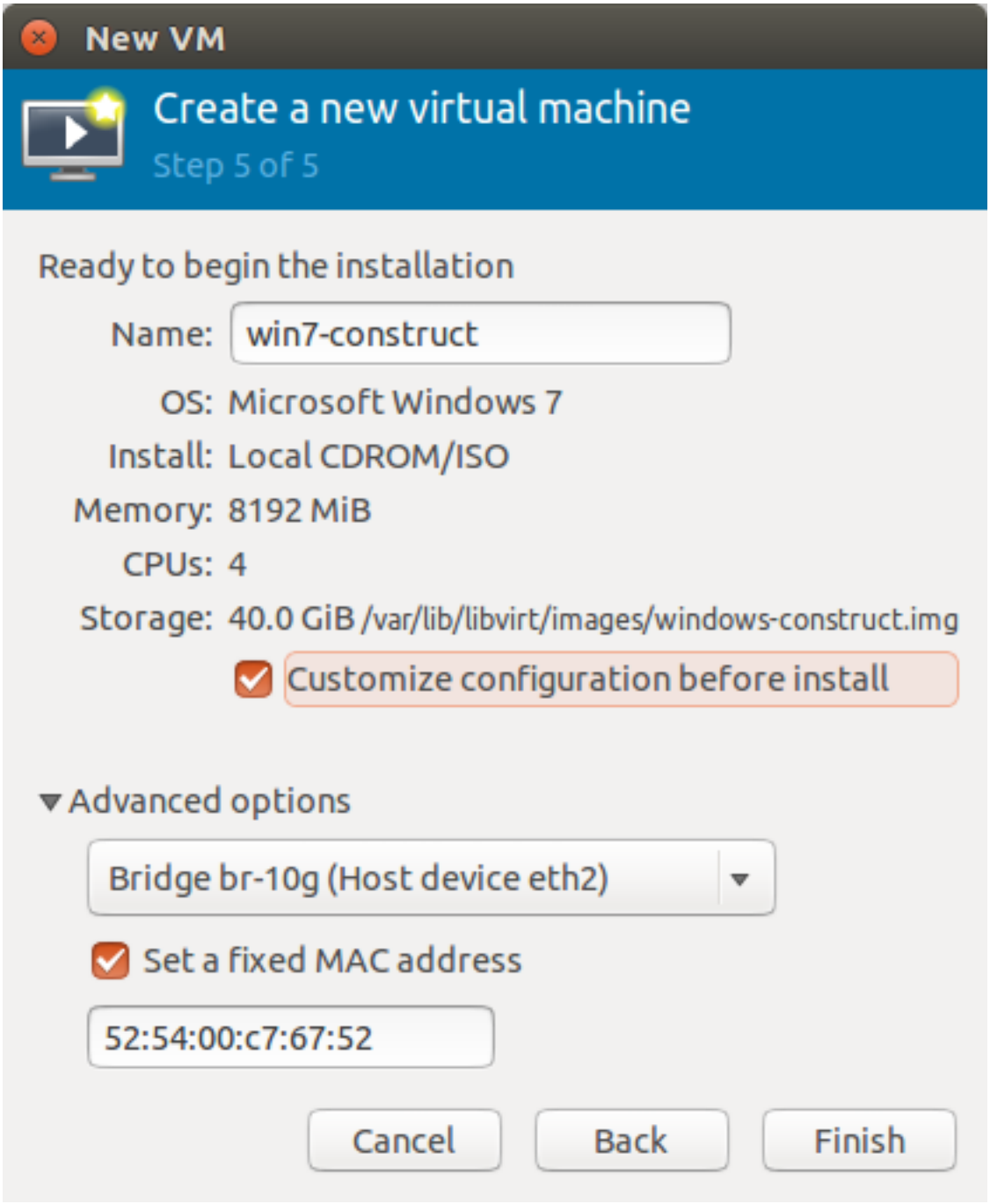
Select New Volume at the resulting screen. We'll be creating a raw volume as it is easier to import into glance than the default qcow2. You'll want at least 50 GB of space to hold all updates; 60 GB is advisable.



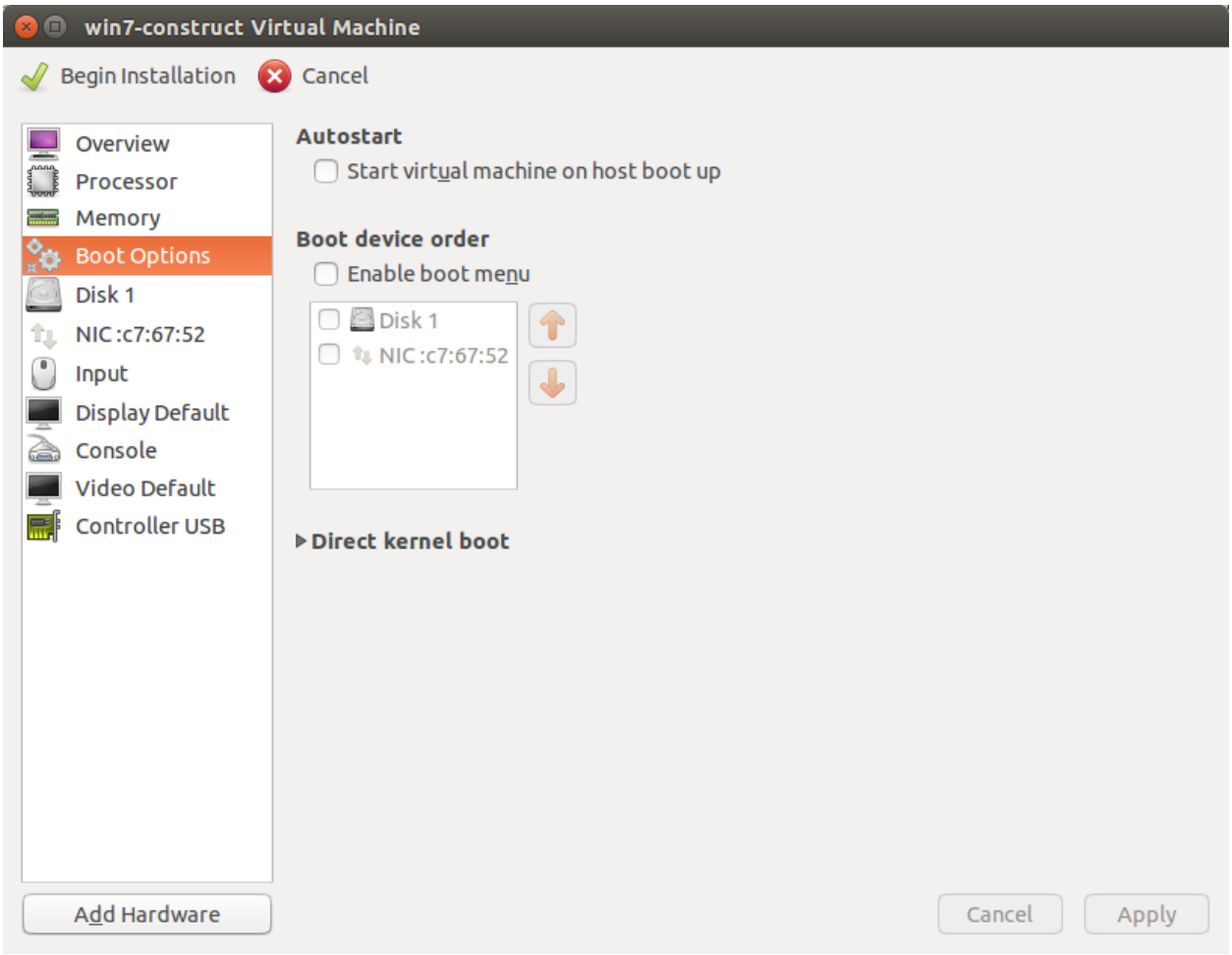
After finishing with the volume, you'll return to the volume selection screen. Select the new volume.



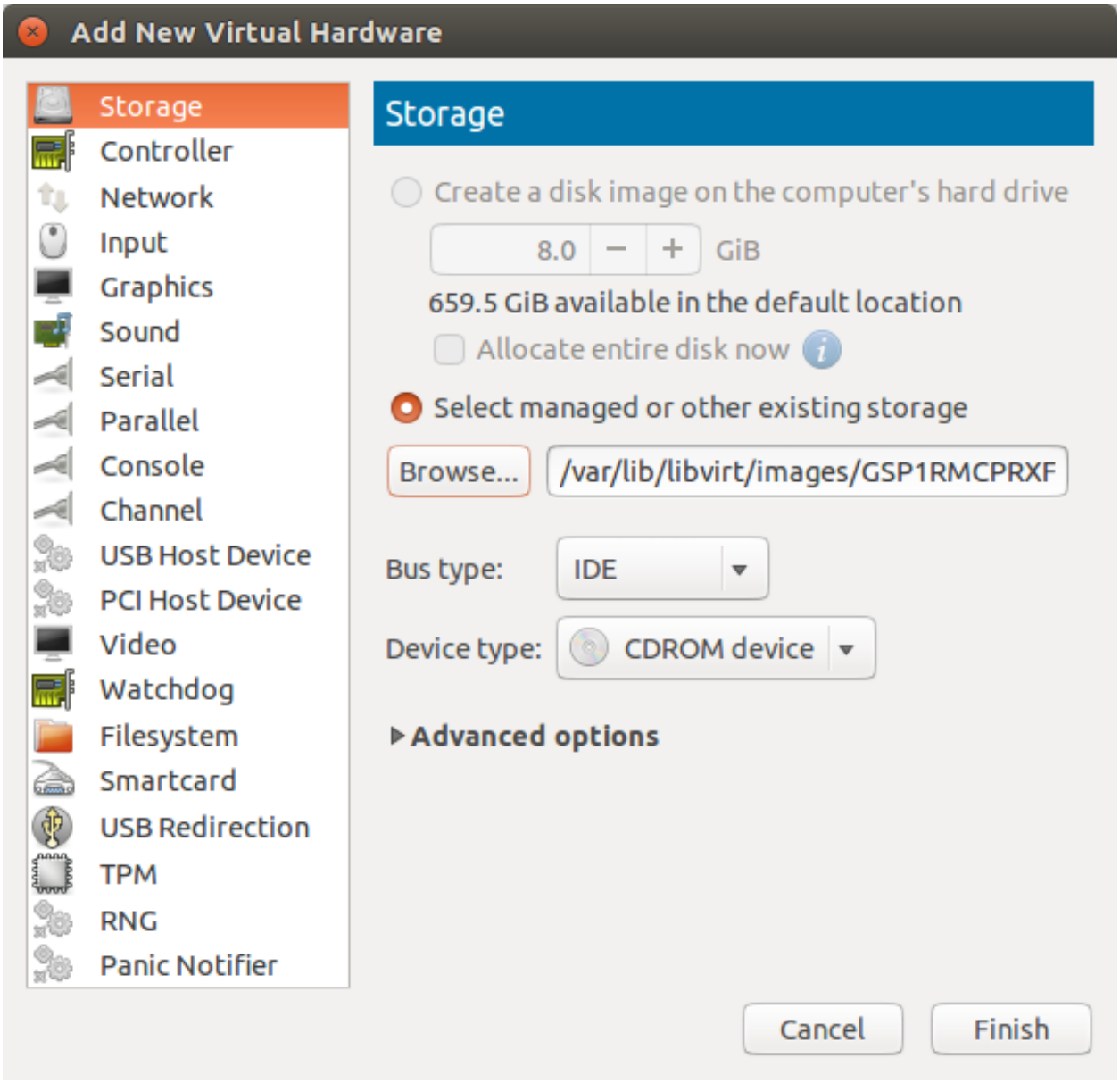
Name the VM. Make sure it is set up on the host's eth2 bridge, and opt to customize configuration before install. Click Finish.



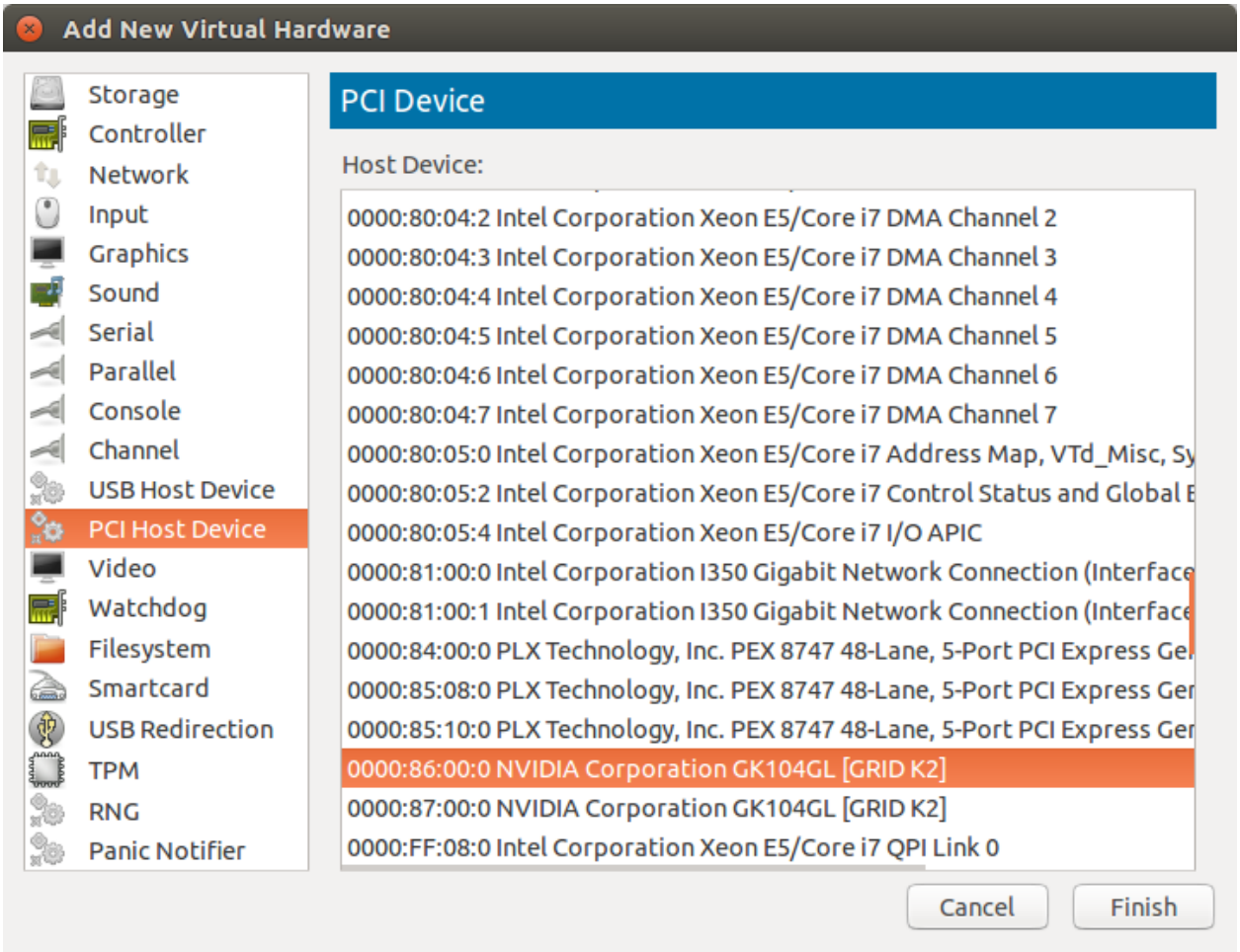
There are several post-config steps. First, click Add Hardware.



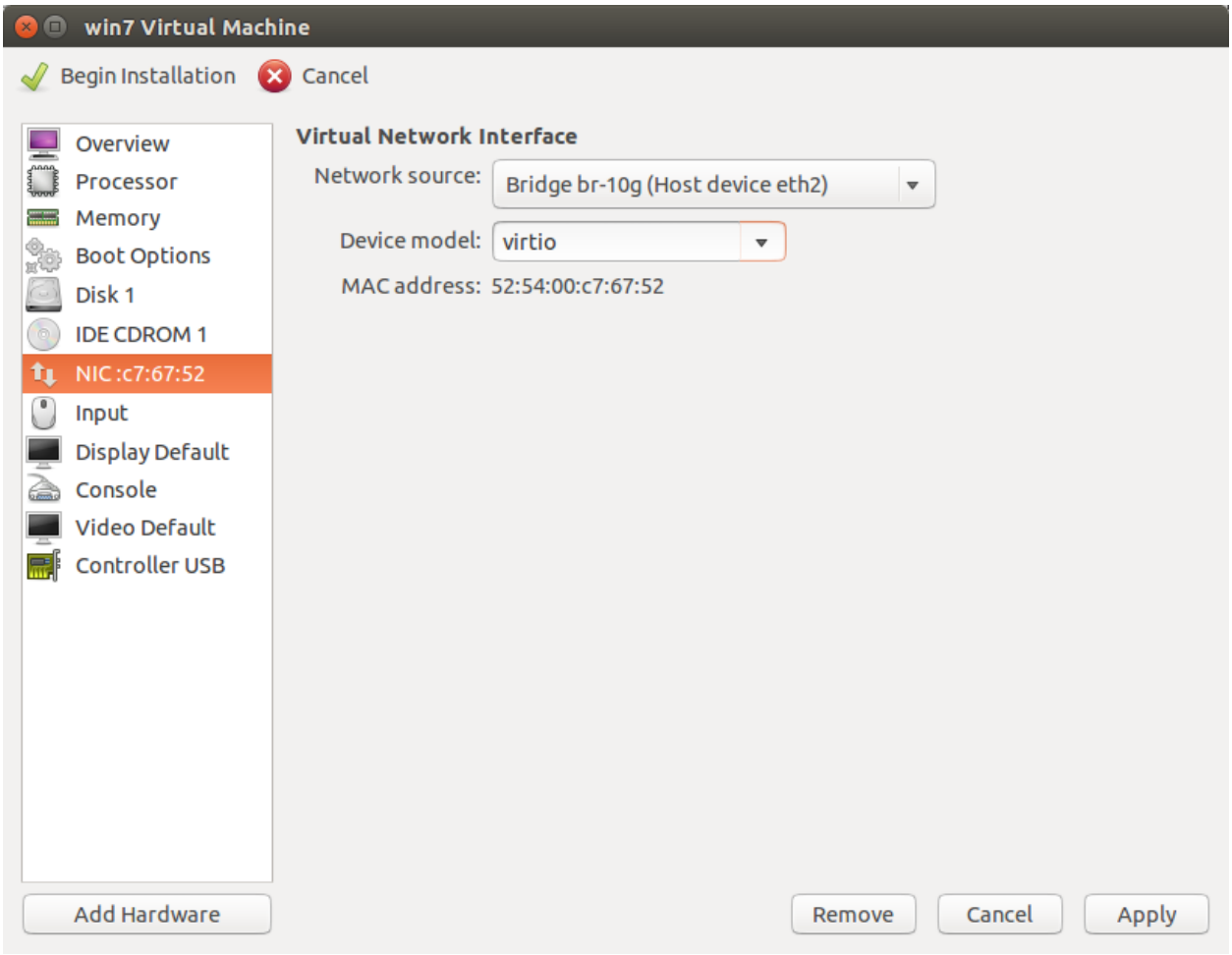
Select the storage option. Select the Windows install ISO from manged storage and ensure that it mounts as a CDROM with an IDE bus. Click Finish.



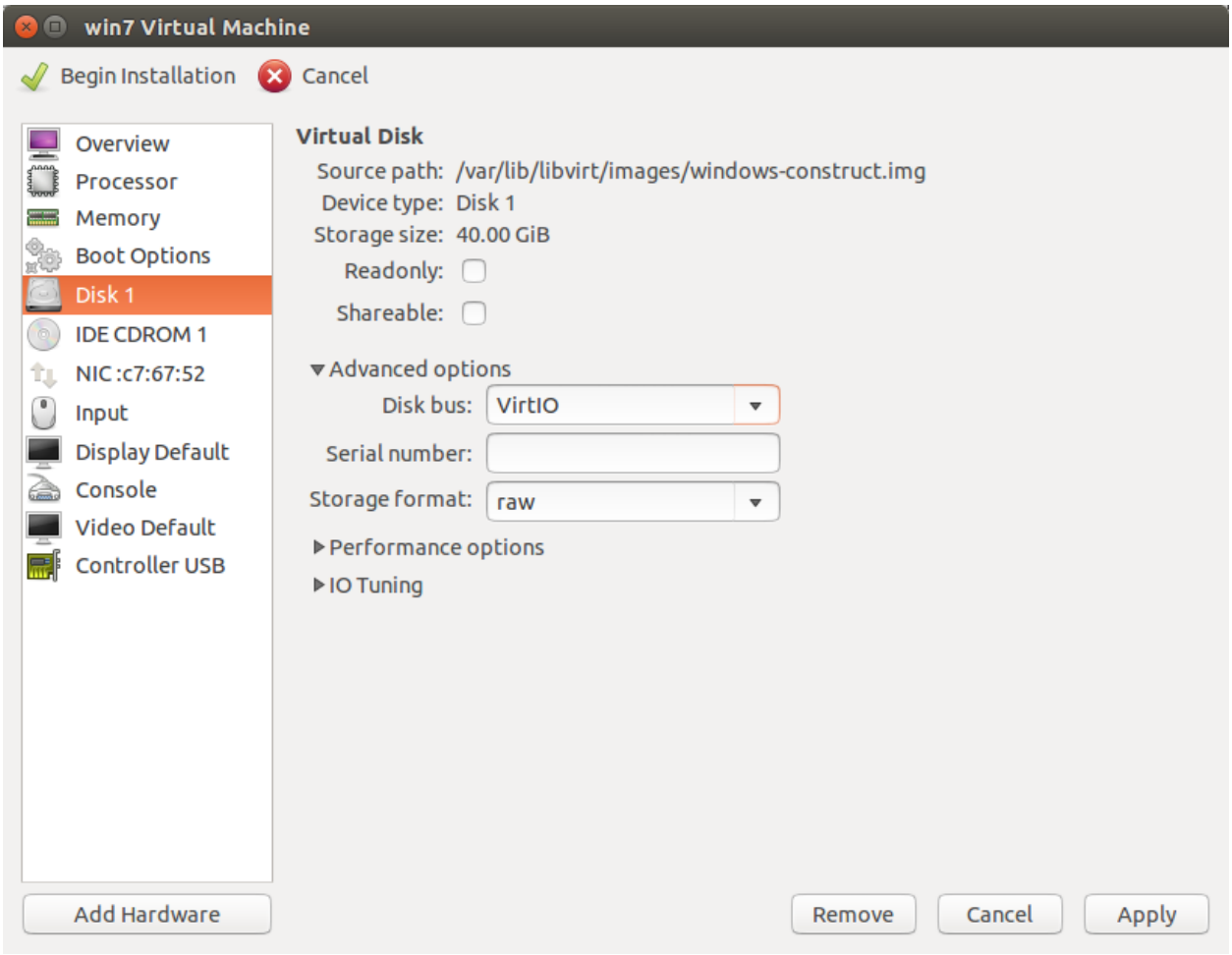
Select Add Hardware again. Select PCI Host Device. Navigate to one of the grid cards and select it. Be very careful that no other VMs are using the card. Click Finish.



Within the list of already configured devices, navigate to the NIC. Set its device model to virtio.

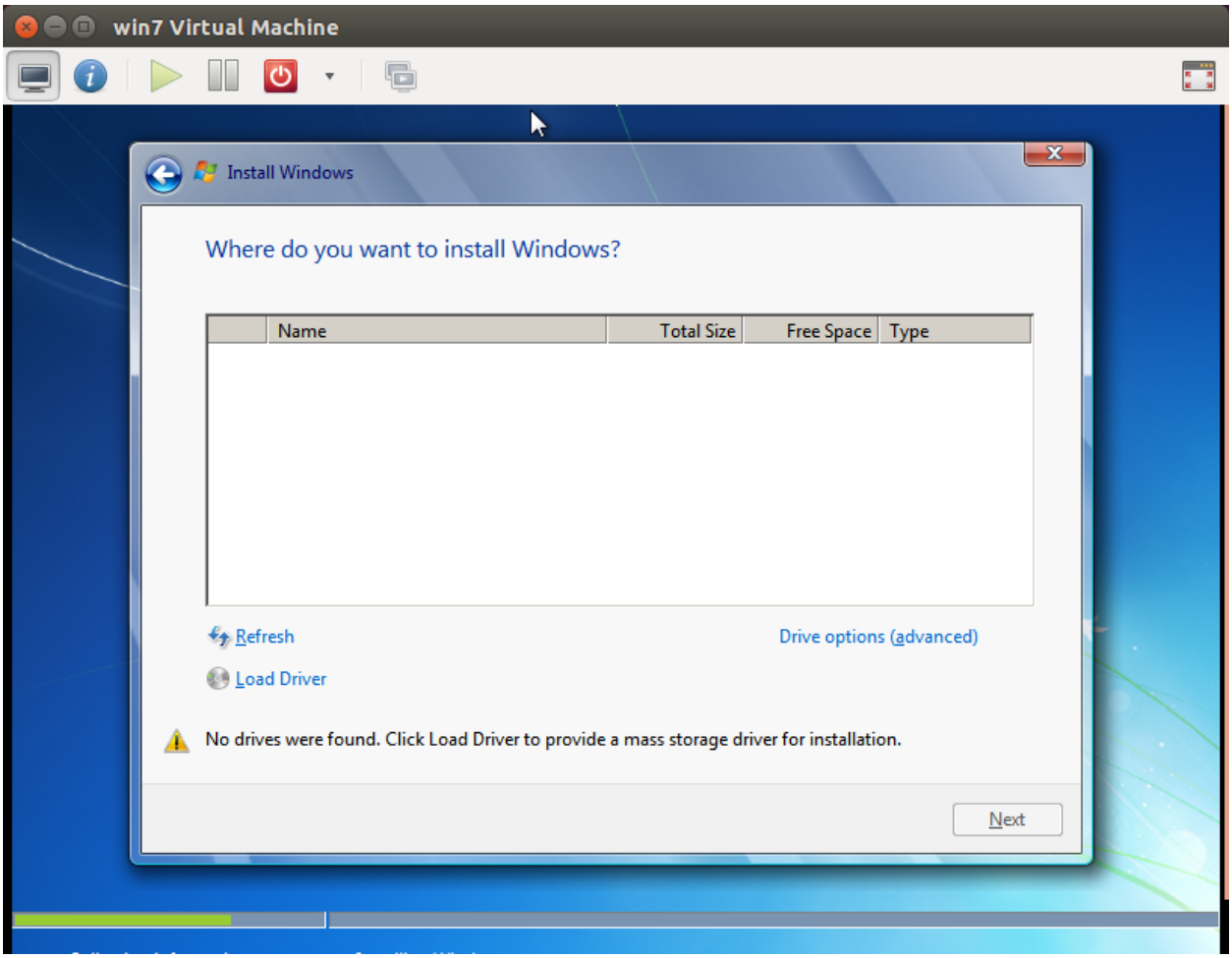


Do similarly for the hard disk. Set its bus to VirtIO. Click Apply and begin installation.

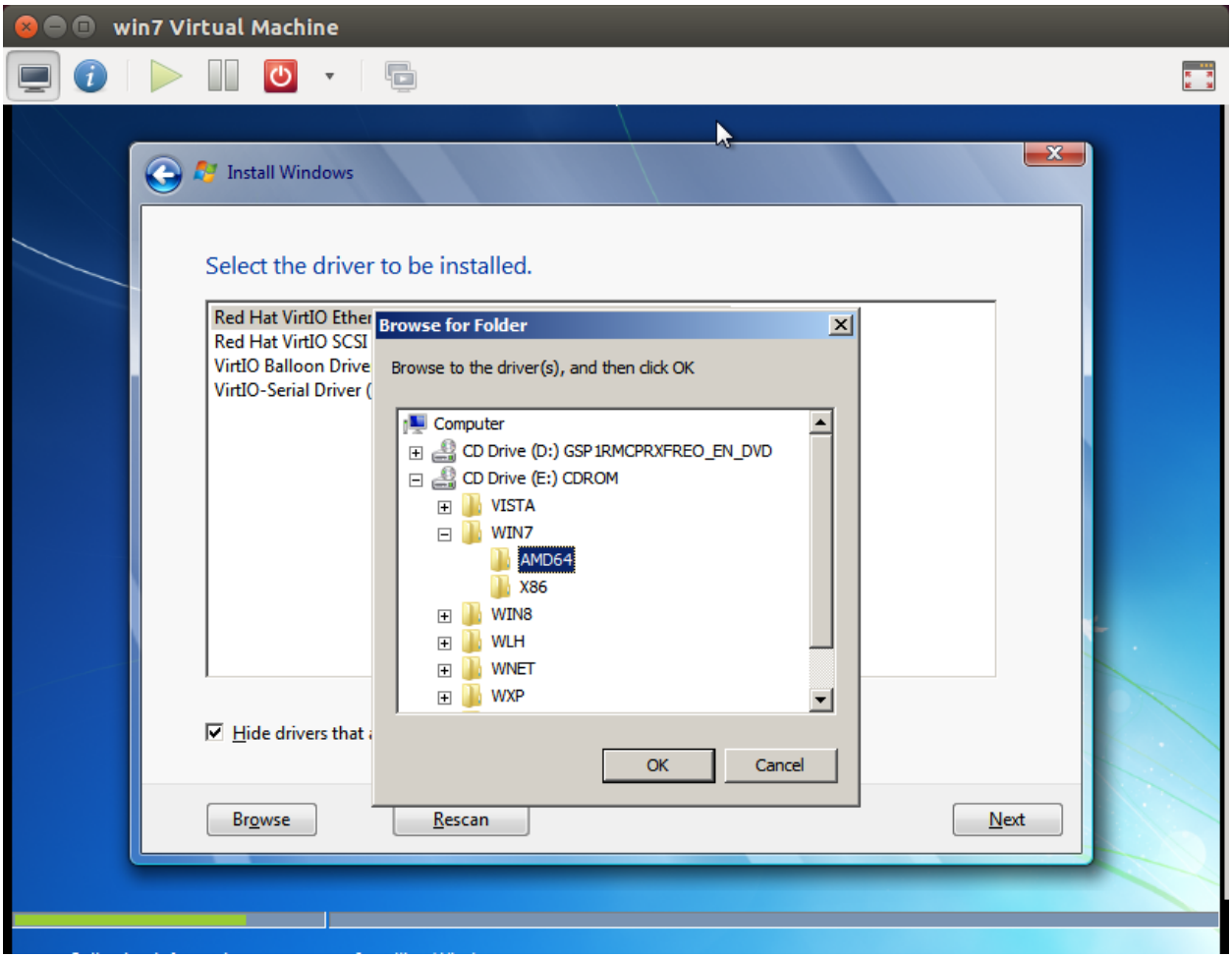


1.13.4 Windows Installation

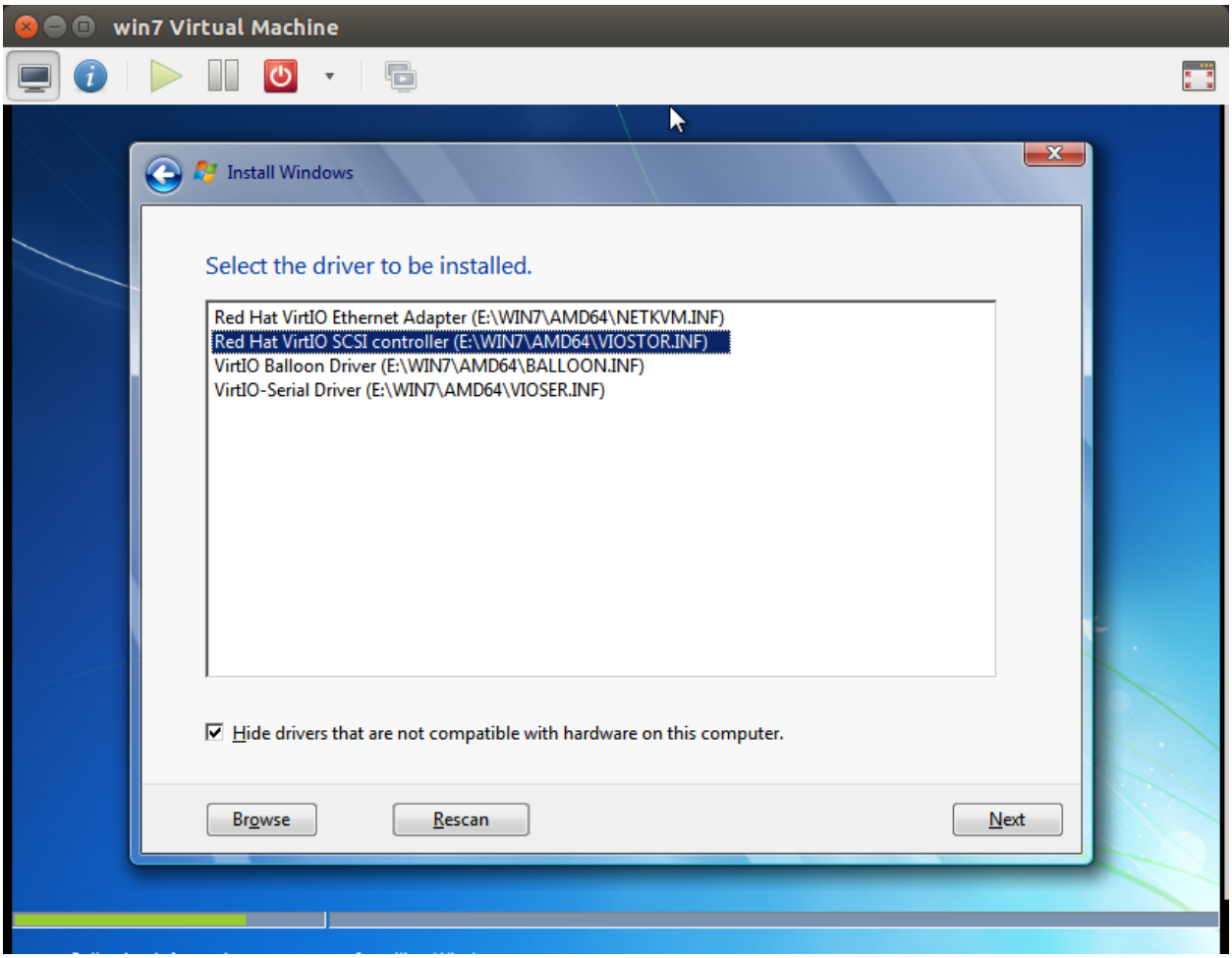
Partway through the Windows installation, when you would be prompted to select a drive, the installer will complain that no drives were found. Here, you will need to load the drivers necessary to see and use the attached virtio drive.



Browse to the virtio CDROM and navigate to the AMD64 directory under the proper version of Windows, i.e. E:\WIN7AMD64. Select it and click OK.

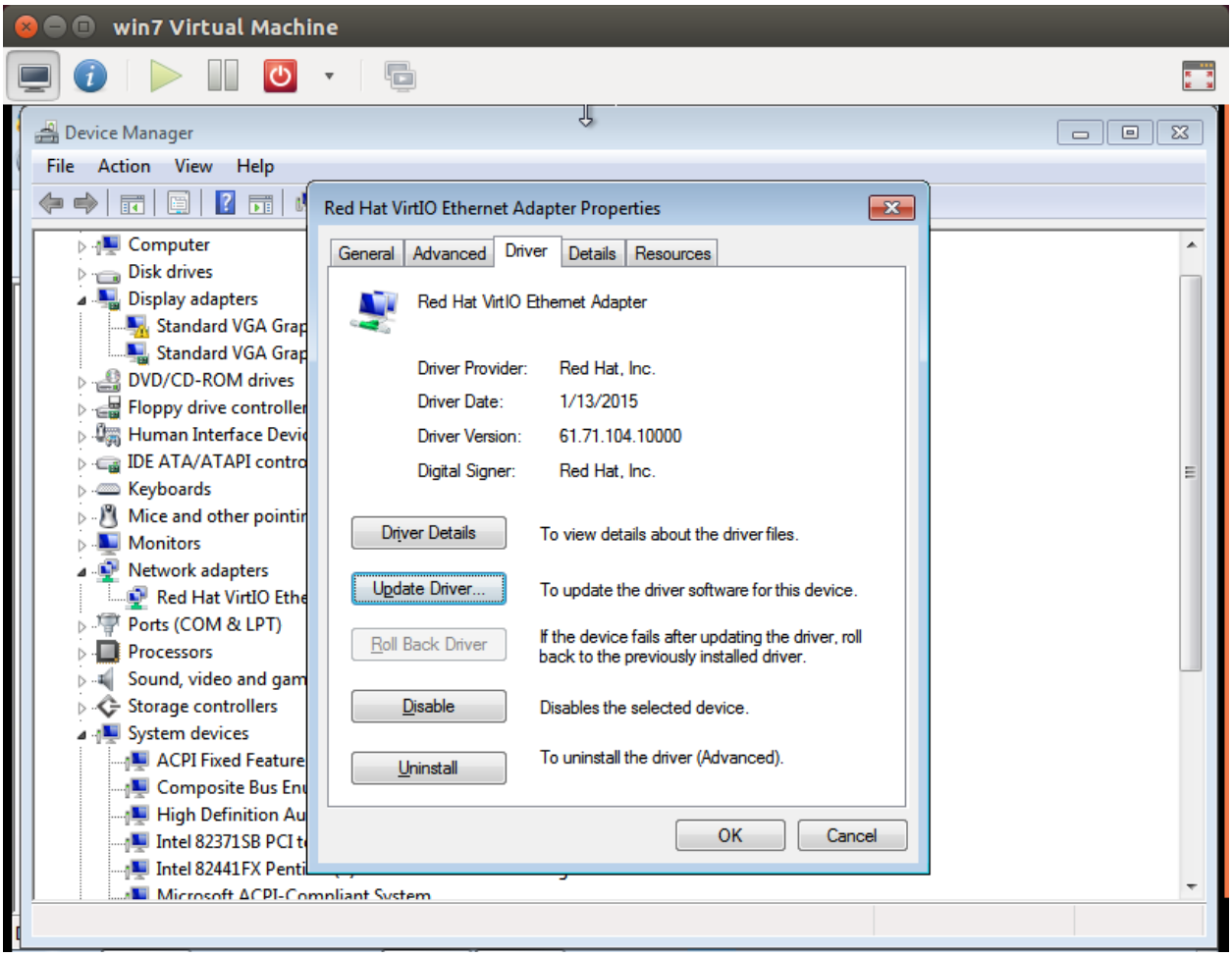


Select the Red Hat VirtIO SCSI controller driver. The disk you created in the initial setup will now appear, and you can continue the rest of the installation normally.

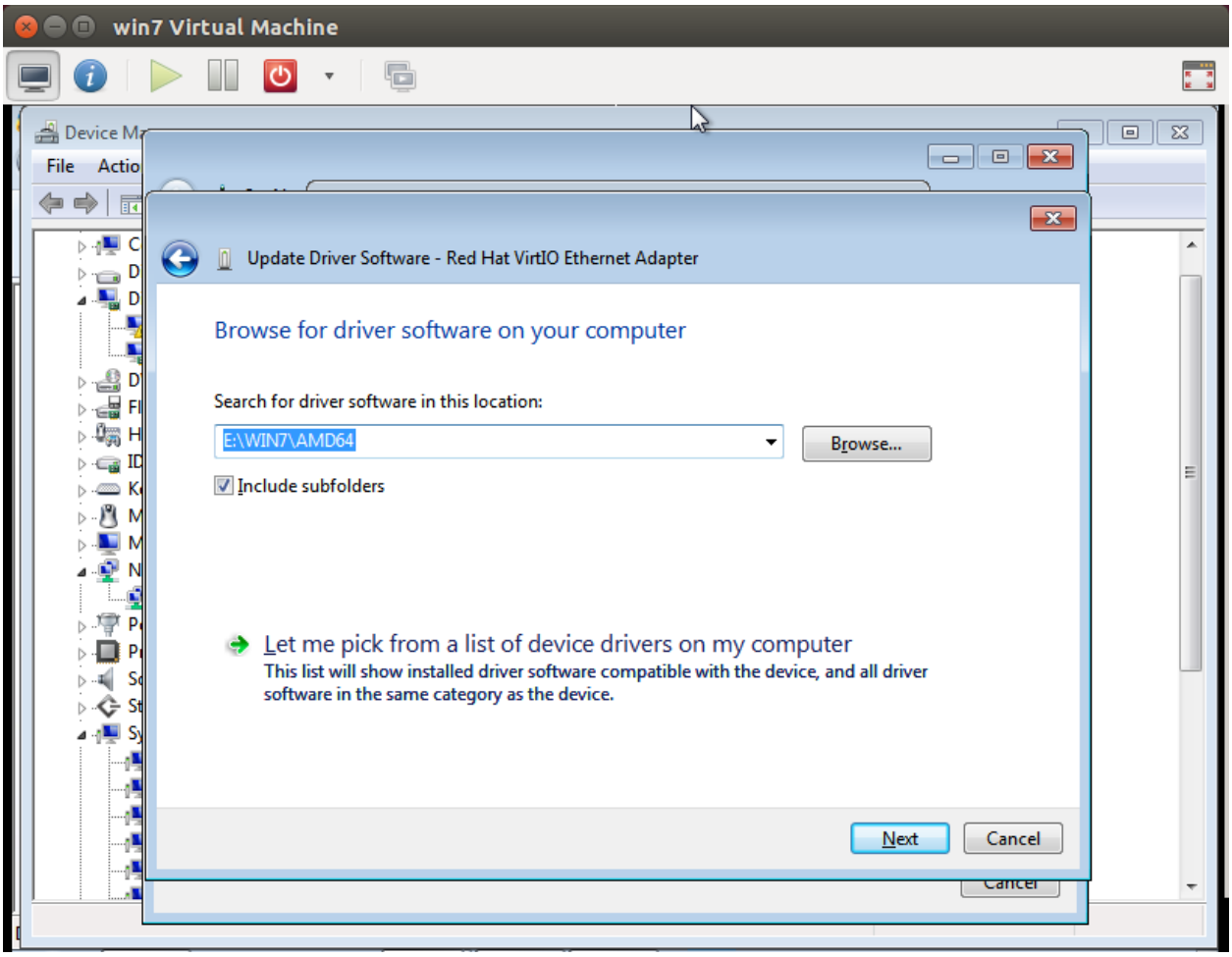


1.13.5 Post-Installation

Log into Windows. Open the Device Manager. Wherever you see a device without a functioning driver which will be marked by a yellow exclamation point, open it unless it is a graphics device.



Choose to browse locally for an appropriate driver. Navigate to the same folder on the driver install disk you selected for the initial install.



Open a command prompt with Administrator privileges. Set a static IP for the default Local Area Connection on the vmhost's bridge with the following command:

```
# netsh interface ip set address name="Local Area Connection"  static  [Instance IP␣
↪Address] [Bridge Netmask] [Bridge default gateway]
```

Configure Windows update and bring your installation up to scratch. This will take a great deal of time.

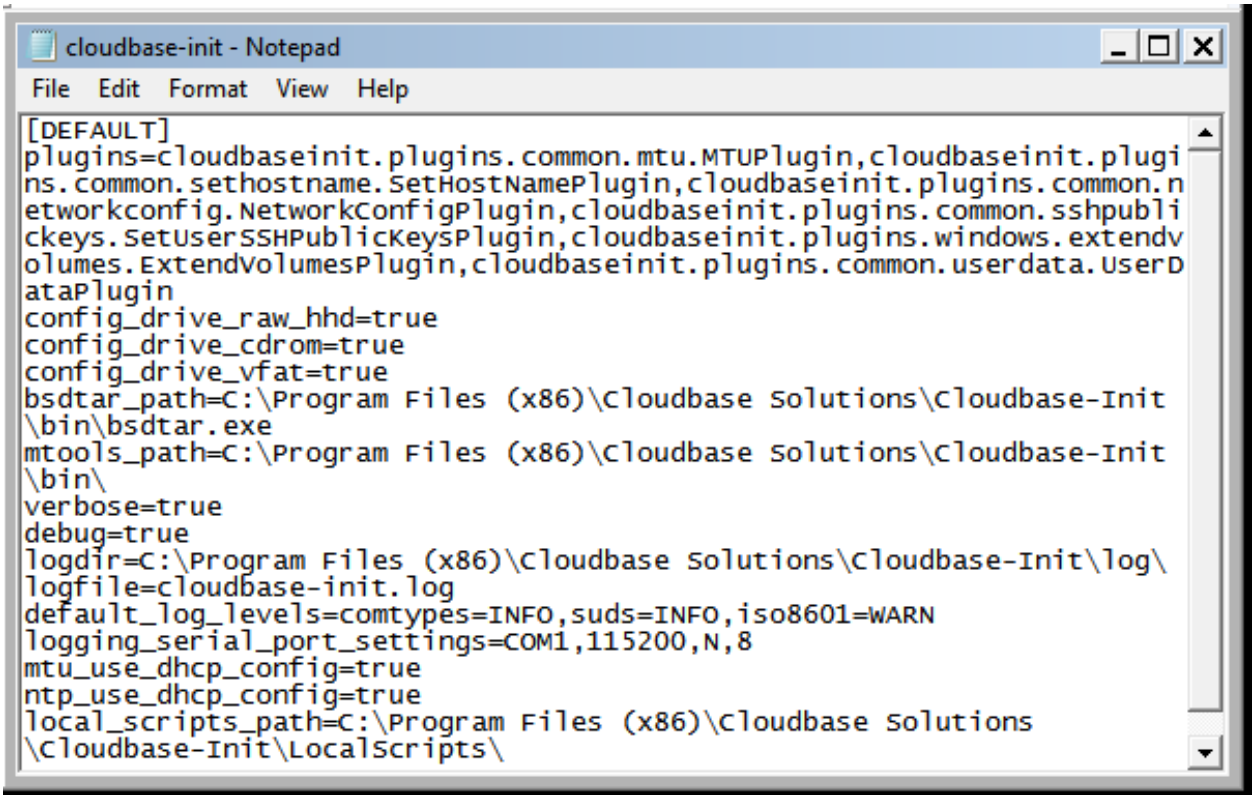
You should now be able to reach the Internet through a web browser.

- Start powershell and run `Set-ExecutionPolicy Unrestricted`.
- Download, install, and configure WinSCP.
- Download Cloudbase's cloud-init for Windows.
- Using WinSCP, download the NVIDIA sdk drivers and the SCW installer from the vmhost.
- Install the latest 1.0.2 version of Win32 OpenSSL - https://slproweb.com/products/Win32OpenSSL.html
- Install SCW.
- Configure SCW using the SCW User Guide - http://www.penguincomputing.com/documentation/scyld-cloud-workstation/user-guide/
- Install the NVIDIA drivers, but postpone restarting.
- Install cloudbase-init.

- When prompted to run sysprep, do so.

Postpone shutting down again.

Open cloudbase-init's configuration directory at `C:\Program Files (x86)\Cloudbase Solutions\Cloudbase-Init\conf`. You'll need to disable the utility's CreateUser and SetPassword plugins. To do so, add a `plugins=` line to both the `cloudbase-init.conf` and `cloudbase-init-unattend.conf` files specifying which plugins are to be loaded and run, excluding those plugins. See the working list in the following screenshot.



1.13.6 Openstack Integration

Shut down the instance from within it, and from the vmhost, copy the VM's backing disk file to the controller via SCP. Save the disk image on the vmhost for later use.

```
root@vmhost3:~# scp /var/lib/libvirt/images/windows-construct.img cc1:/root/
```

Import the disk file into glance on the controller.

```
# glance image-create \
    --name WindowsUpdatedLoginNode \
    --disk-format raw \
    --container-format bare \
    --min-disk 80 \
    --min-ram 8192 \
    --file ./win-construct.img \
    --is-public True \
    --progress \
    --property active=true \
```

(continues on next page)

(continued from previous page)

```
    --property image_type=loginnode \
    --property service_url='https://{}/' \
    --property hw_video_model=qxl
```

Create a cinder volume from the resultant glance image.

```
# cinder create \
    --image-id $(glance image-list | awk '/WindowsUpdatedLoginNode/ {print $2}') \
    --display-name "WindowsLoginVolume" \
    80
```

If you want to use RDP as well as SCW, make sure the SCM cloud controllers have the *openstack.allow_rdp* setting as True.

Ensure that the flavor you want can contain the volume you created, and if using GPU acceleration, that it asks for a graphics card.

```
# nova flavor-show 8
+----------------------------+-----------------------------------------+
| Property                   | Value                                   |
+----------------------------+-----------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                   |
| OS-FLV-EXT-DATA:ephemeral  | 0                                       |
| disk                       | 40                                      |
| extra_specs                | {"pci_passthrough:alias": "K2_Grid:1[a]"} |
| id                         | 8                                       |
| name                       | scm-k2.4x1                              |
| os-flavor-access:is_public | True                                    |
| ram                        | 1024                                    |
| rxtx_factor                | 1.0                                     |
| swap                       |                                         |
| vcpus                      | 4                                       |
+----------------------------+-----------------------------------------+
```

Boot an instance from the volume you created above. Be sure to specify a keypair.

```
# nova boot \
   --flavor 8 \
   --key-name default \
   --meta admin_pass=Penguin \
   --block-device-mapping \
      vda=$(cinder list | awk '/WindowsLoginVolume/ {print $2}'):::0 \
   Windows_Instance
```

Associate a floating IP with the instance if one is not automatically assigned.

```
# openstack floating ip create

# openstack server add floating ip Windows_instance 10.110.20.32
```

Connect via SCW or RDP if enabled. The password you used during install should have persisted. Verify that the instance is aware of the card by opening Device Manager. If everything is working at this point, you're good to go.

1.13.7 Creating New Login Node Templates

Unlike with Centos Login Nodes, we can't run `virt-sysprep` and similar utilities on Windows disk images, so our usual routine of exporting the cinder volume of an instance where we've made the changes will not work. Instead, save the initial libvirt disk-image you created in the initial installation. If you need to make any changes that you wish to appear on all spawned instances, spin up the libvirt instance in virt-manager in safe-mode, make the changes, shut down, and repeat the import steps detailed in the previous sections.

1.14 Exporting and Prepping a VM disk image for OpenStack

If you have a running virtual machine in OpenStack and want to export it, shrink it, and prep it for installation to Glance to use as a template for other machines, use these instructions.

Note: In order to get `cloud cloud-init` working with `resizefs` in CentOS 6.x, you need to install it manually in the VM.

Instructions here:.

Short version:

Install `cloud-init` packages and `git` (this one is required to install `linux-rootfs-resize`):

```
# yum install -y cloud-utils cloud-init parted git
```

Install linux rootfs resize:

```
# cd /tmp
# git clone https://github.com/flegmatik/linux-rootfs-resize.git
# cd linux-rootfs-resize
# ./install
```

Note: This has to be done each time the kernel is updated.

Examples of when this needs to be done:

- Creating a new head node image
- Creating a new login node image

You will want to install your packages in a VM that was "boot from volume", so that there is a persistent volume in Cinder left around after you power off the VM.

In this example, we will grab the disk called `loginnode image`:

```
# cinder list
+--------------------------------------+--------+----------------------+------+-------
↪------+----------+--------------------------------------+
|                  ID                  | Status |         Display Name
↪ | Size | Volume Type | Bootable |                 Attached to                  |
+--------------------------------------+--------+----------------------+------+-------
↪------+----------+--------------------------------------+
| 51e27df2-387b-40f1-97e1-50a848707fb1 | in-use |   loginnode image    |  10  |
↪      None        |   true   | 91da14b6-fbf3-4fa0-b4b5-0c4cd504a9ca |
```

Note the ID.

You will then need to export this disk onto a machine to work with. This machine will need access to ceph (i.e. `rbd ls` works), and will need the following tools:

```
libguestfs-tools
guestmount
guestfish
```

Export the image:

```
# rbd export volumes/volume-51e27df2-387b-40f1-97e1-50a848707fb1 loginnode.raw
```

Prep the image (this removes log files, persistent net rules, etc):

```
# virt-sysprep -a loginnode.raw
```

Use `guestfish` to resize the root filesystem down to its minimum size. In this example, we know that `/dev/vda1` is `/boot`, and `/dev/vda2` is `/`.

```
# guestfish --rw -a loginnode.raw


Welcome to guestfish, the libguestfs filesystem interactive shell for
editing virtual machine filesystems.


Type: 'help' for help on commands
          'man' to read the manual
          'quit' to quit the shell


><fs> run
><fs> list-filesystems
/dev/vda1: ext4
/dev/vda2: ext4
><fs> e2fsck-f /dev/vda1
><fs> e2fsck-f /dev/vda2
><fs> resize2fs-M /dev/vda2
```

Now, check the size of the resulting file system:

```
><fs> mount /dev/vda2 /
><fs> mount /dev/vda1 /boot
><fs> df-h
Filesystem          Size  Used Avail Use% Mounted on
tmpfs                96M  152K   96M   1% /run
/dev/vda2           2.1G  2.1G         0 100% /sysroot
/dev/vda1           481M   99M  354M  22% /sysroot/boot


><fs> quit
```

The two devices use just over 2.5G. We will need to know this.

Now, sparsify the disk (fill what we can with zeroes):

```
# virt-sparsify loginnode.raw loginnode.sparse.raw

Create overlay file to protect source disk ...

Examine source disk ...
Fill free space in /dev/vda1 with zero ...
Fill free space in /dev/vda2 with zero ...
Copy to destination and make sparse ...


Sparsify operation completed with no errors.  Before deleting the old
disk, carefully check that the target disk boots and works correctly.
```

Even though we minimized the root filesystem, we haven't shrunk the partition inside the disk. You can see that with:

```
# virt-filesystems --long --parts --blkdevs -h -a loginnode.sparse.raw
Name           Type           MBR  Size  Parent
/dev/sda1  partition  83   500M  /dev/sda
/dev/sda2  partition  83   9.5G  /dev/sda
/dev/sda   device         -        10G   -
```

Resize the `/dev/sda2` partition. Create a new disk (not a partition) that will hold everything. Since we have 2.5G of files, we will create a 3G disk:

```
# truncate -s 3G[b] outdisk
# virt-resize --shrink /dev/sda2 loginnode.sparse.raw outdisk
Examining loginnode.sparse.raw ...
**********


Summary of changes:


/dev/sda1: This partition will be left alone.


/dev/sda2: This partition will be resized from 9.5G to 2.5G.  The
        filesystem ext4 on /dev/sda2 will be expanded using the 'resize2fs'
        method.


**********
Setting up initial partition table on outdisk ...
Copying /dev/sda1 ...
Copying /dev/sda2 ...
Expanding /dev/sda2 using the 'resize2fs' method ...


Resize operation completed with no errors.  Before deleting the old
disk, carefully check that the resized disk boots and works correctly.
```

Now `outdisk` has our finished disk file, and can be renamed and imported into Glance. You can check the final partition size if you like.

```
# virt-filesystems --long --parts --blkdevs -h -a outdisk
Name           Type           MBR  Size  Parent
/dev/sda1  partition  83   500M  /dev/sda
```

(continues on next page)

(continued from previous page)

```
/dev/sda2  partition  83   2.5G  /dev/sda
/dev/sda   device         -        3.0G  -
```

1.14.1 Importing to OpenStack

If you are importing a new "loginnode image" follow these steps.

First, deactivate the old image so it's not used anymore:

```
# glance image-update --name Login_Node_retired --property active=false Login_Node
```

Upload the new resized image to glance and activate with:

```
# glance image-create \
    --name Login_Node \
    --disk-format raw \
     --container-format bare \
     --min-disk 10 \
     --min-ram 512 \
     --file ./outdisk \
     --is-public True \
     --progress \
     --property active=true \
     --property image_type=loginnode
```

1.15 GPU Passthrough for KVM

To use GPU hardware with OpenStack, KVM, and SCM, you need to make some manual changes to the default configurations. More details are available in the OpenStack document Pci passthrough. This document uses the nVidia K2 Grid card in examples.

1.15.1 nova.conf

Edit `/etc/nova/nova.conf`. You'll need the `product_id` and `vendor_id` of your GPU. You can get these from the output of the `lspci` command, or a site like The PCI ID Repository or PCIDatabase.com. An example, using the Nvidia K2 card:

```
pci_alias = { 'name': 'K2_Grid', 'vendor_id': '10de', 'product_id': '11bf' }
pci_passthrough_whitelist = [ { 'vendor_id': '10de', 'product_id': '11bf' } ]
```

Also create a section for `spice` and enable it:

```
[spice]
enabled = True
```

1.15.2 nova-compute.conf

Add the `spice` section to `/etc/nova/nova-compute.conf` also:

```
[spice]
enabled = True
```

1.15.3 Flavor changes

Specify the maximum number of megabytes of video RAM and enable device passthrough with these flavor *extra_spec*s:

```
hw_video:ram_max_mb=64
"pci_passthrough:alias"="K2_Grid:1"
gpu=true
```

Do this by updating an existing flavor with `nova flavor-key`. Example:

```
# nova flavor-key my_existing_flavor set \
    hw_video:ram_max_mb=64 \
    "pci_passthrough:alias"="K2_Grid:1" \
    gpu=true
```

The first value on the right side the second item above (`K2_Grid`) should match the value of *name* in the *pci_alias* entry for `nova.conf` above. The second value (`1`) is the number of items passed through.

1.15.4 Image changes

For the GPU, set these *properties* on your image:

```
hw_video_model=qxl
hw_video_ram=8
configure_x=true
```

`configure_x` is only used by Linux GPU images; it gets cloud-init to generate an `Xorg.conf` file with the device's PCI address.

Do this when you create the image with `glance image-create`. Example:

```
# glance image-create \
    --name my_new_image  \
    --is-public true \
    --file /root/images/my_raw_image.raw
    --disk-format raw \
    --container-format bare \
    --min-disk 15 \
    --min-ram 512
    --progress \
    --property image_type=loginnode \
    --property service_type=loginNode \
    --property hw_video_model=qxl \
    --property hw_video_ram=8 \
    --property configure_x=true \
    --property service_url='https://{}/'
```

You can also set these properties when you update an existing image with `glance image-update`:

```
# glance image-update \
    --property hw_video_model=qxl \
    --property hw_video_ram=8   \
    --property configure_x=true   \
    my_existing_image
```

The `image_type` and `service_type` properties are used by SCM to allow the image to be displayed in the Scyld Cloud Portal.

SCM needs to know when the image has booted into an active VM. The default is to attempt to ssh to the VM until it responds or times out. The `service_url` property is used if the image does not support ssh (such as Windows) or does not start the ssh server at boot time, but does have a web server that starts at boot time. The value is the URL of some page of this web server, typically the home page. The `{}` characters are changed to the IP of the VM as it boots, and attempts are made to access that web page.

1.15.5 Aggregates

In OpenStack Nova, a *host aggregate* is a group of hosts with similar metadata (key-value pairs). A host may belong to more than one aggregate. We'll use this to help manage GPU and non-GPU hosts.

Edit `/etc/nova/nova.conf` again and add the `AggregateInstanceExtraSpecsFilter` to `scheduler_default_filters`:

```
scheduler_default_filters=AggregateInstanceExtraSpecsFilter,RetryFilter,
↪AvailabilityZoneFilter,RamFilter,ComputeFilter
```

Create three aggregates, the first being the availability zone:

```
# nova aggregate-create aggregate_Denver Denver
# nova aggregate-create GPU Denver
# nova aggregate-create nonGPU Denver
```

Next, set metadata on the aggregates:

```
# nova aggregate-set-metadata GPU gpu=true
# nova aggregate-set-metadata nonGPU gpu=false
```

Then add the host to the aggregate:

```
# nova aggregate-add-host nonGPU vmhost1
# nova aggregate-add-host nonGPU vmhost2
# nova aggregate-add-host GPU vmhost3
```

Next, set the matching metadata on the nova flavors:

```
# nova flavor-key 1 set gpu=false
# nova flavor-key 2 set gpu=false
# nova flavor-key 3 set gpu=false
# nova flavor-key 4 set gpu=true
# nova flavor-key 5 set gpu=true
```

Instead of omitting the `gpu` metadata for the non-GPU flavors, we're explicitly setting them to `false`.

1.15.6 Cloud Controller

The Scyld Cloud Controller can check for GPU usage before starting a SCM VM.

In the OpenStack Galera database, accessible through Docker on one of the OpenStack controller systems, add a user with SELECT permissions for the nova database.:

```
# ssh cc1
# docker exec -it mariadb /bin/bash
# mysql -u root -pOPENSTACK_ROOT_DB_PASSWORD
# CREATE USER 'cc'@'%' IDENTIFIED BY 'SECURE_PASSWORD';
# GRANT SELECT ON nova.pci_devices TO 'cc'@'%';
```

On the Scyld Cloud Controller server, edit the `cloudcontroller.ini` file, and add the `openstack.gpu_query` setting, consisting of an URL string for connecting to the OpenStack database and user created.:

```
openstack.gpu_query = mysql://cc:SECURE_PASSWORD@openstack_host
```

Next, add the `product_id` of the GPUs being used as a space-separated list to `openstack.gpu_pci_ids`:

```
openstack.gpu_pci_ids = 0ff2 11bf 13f2
```

Touch the Scyld Cloud Controller's wsgi file to make the change active.:

```
touch /var/www/wsgi/cloudcontroller.wsgi
```

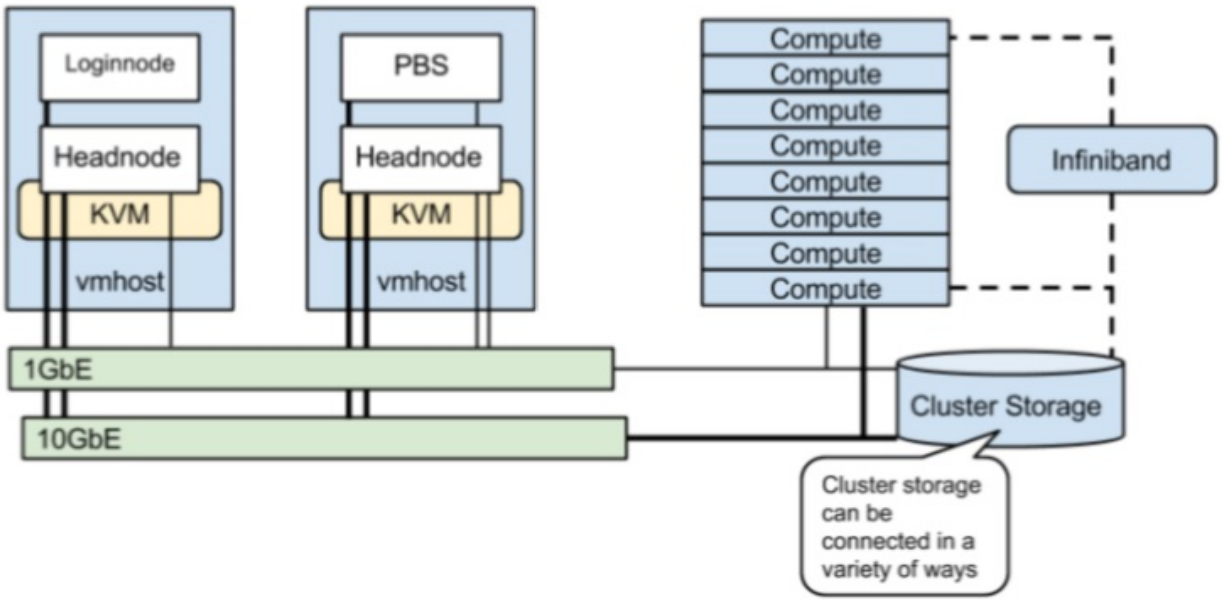
SCM VMs that require GPUs will now show an error message when all available GPUs are in use.

1.16 Cluster Integration

SCM can be used to run virtual machines for Scyld ClusterWare, Penguin Computing's cluster management software suite. ClusterWare is traditionally run on physical nodes, often with one single node acting as the master node. Running ClusterWare on virtual nodes presents a lot of advantages:

- Ability to "right-size" the head node. No wasted capacity or propensity to over-provision.
- Ability to live-migrate the head node when the host needs maintenance. Less down time.
- Ability to easily snap-shot the head node disk. Backups can be exported and stored, or snapshots can be rolled back to.
- Easier to run multiple head nodes in "multi-master" mode. No need to purchase additional hardware for a second head node.

In addition to running multiple head nodes, SCM makes it easier to provision additional management nodes for things like schedulers (i.e. Maui or MOAB). Here is a figure that shows a typical SCM deployment with ClusterWare:



This diagram does not show the SCM controller nodes, only the VM hosts. In order to protect the head nodes against VM host failure, it is important to ensure that each head node is on a different VM host.

1.17 Shibboleth as Service Provider

Shibboleth can be used as the Service Provider in a SAML Single Sign-On (SSO) environment for the Scyld Cloud Portal.

SSO logins use the following attributes from the SAML payload, as set in attribute-map.xml and identified by the id value:

username fullName or firstName, lastName email

Example:

```
<Attribute name="urn:oid:2.16.840.1.113730.3.1.241" id="fullName"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.1" id="username"/>
<Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="email"/>
```

The Attribute element may have attributes of name, xmlns:saml and nameFormat depending on the Identity Provider.

1.18 Mariadb in a Galera Cluster Maintenance and Recovery

1.18.1 Introduction

This document covers how to perform system maintenance with the Mariadb database in active production, and how to recover from power outages, or network failure.

1.18.2 Environment

SCM (Scyld Cloud Manager) currently leverages the Kolla OpenStack project which packages the OpenStack services into Docker containers.

The mariadb database utilizes galera to run a database cluster on the three OpenStack controller systems. The cluster provides high availability as well as scalability. The three instances of the database run inside docker containers. Containerization changes how the user interacts with the database, as it differs from running the database directly on bare metal.

The best method of accessing the database is to install mariadb on the ops system which has access to the private OpenStack API network. Several scripts are provided to interrogate the status of the OpenStack cluster as well as the database.

Note: The scripts referenced in this document are shipped with SCM and are installed in the directory `/opt/scm/scripts/`.

1.18.3 Checking Galera Cluster Status

The script `galeraStatus.sh` will report the current status of the database cluster.

Since each of the OpenStack controller systems has an individual copy of the database for redundancy, the Galera software ensures that the three copies of the database are kept in sync with each other.

To protect the integrity of the data, Galera requires a quorum or majority of controller instances to agree on the last update to the database. In the SCM case there are three controllers, a minimum of two controllers must agree on the last committed transaction. This means that at least two instances of the database must be running and communicating to agree on every update.

Without quorum Galera will not allow updates to be done ensuring that the instances are kept synchronized and avoid a condition called split-brain.

A healthy cluster will have a similar report from running the `galeraStatus.sh` script.

Variable_name	Value
wsrep_last_committed	3832840
wsrep_local_state_comment	Synced
wsrep_incoming_addresses	10.10.3.5:3306,10.10.3.1:3 306,10.10.3.3:3306
wsrep_cluster_size	3
wsrep_cluster_status	Primary
wsrep_ready	ON

All the fields in this report are important.

- wsrep_last_committed: An ever increasing transaction ID.
- wsrep_local_state_comment: The state of the cluster. **Synced** is the healthy state. Possible other states: Joining, Waiting on SST, Joined, Synced or Donor.
- wsrep_incoming_addresses: The connected servers in the cluster; Healthy is **all three**.
- wsrep_cluster_size: Number of servers in the cluster; Again healthy is **3**.
- wsrep_cluster_status: **Primary** means part of quorum and accepting updates; anything else is bad.
- wsrep_ready: **ON** means communicating with an active cluster. OFF means not accepting queries.

Further details can be found on the Galera website at: http://galeracluster.com/documentation-webpages/monitoringthecluster.html

1.18.4 Normal System Maintenance

Since the Galera wrapped Mariadb database is able to continue in production with a quorum of instances (2 out of 3) , maintenance can be performed without downtime. However, this maintenance needs to be done carefully to keep the quorum when performing maintenance.

The recommended way to temporarily remove a controller system from the cluster is to first stop the mariadb container on the one controller, then stop the docker service on that same controller.

Manually stopping the docker service before a system shutdown will reduce recovery time when the system reboots. Relying on the shutdown command to stop all the docker containers, may result in unclean shutdown of docker containers due to a systemd timeout.

For the smallest possible cluster scenario, there will be three systems which will play the roles of controllers, vmhost and ceph data hosts. This scenario currently only exists in non-production development clusters.

Note: If you do not have osd's running on the controller, skip step 2.

```
# docker stop mariadb
# docker exec -t ceph_mon ceph osd set noout # only when osd is on this controller!!
# systemctl stop docker
```

A check of the galera status with one mariadb database shutdown will resemble the following report:

Variable_name	Value
wsrep_last_committed	3829914
wsrep_local_state_comment	Synced
wsrep_incoming_addresses	10.10.3.1:3306,10.10.3.3:3 306
wsrep_cluster_size	2
wsrep_cluster_status	Primary
wsrep_ready	ON

The controller with the shutdown docker can now have its maintenance performed and then be rebooted.

1.18.4.1 Restarting a Controller

Once the controller is booted back into service, docker will automatically start the containers except for the mariadb container which was manually stopped. Verify all docker containers, with the exception of the manually stopped mariadb, have started and have a status of **Up** using the command:

```
# docker ps -a
```

If the noout flag was previously set in ceph, unset that flag now using the command:

```
# docker exec -t ceph_mon ceph osd unset noout
```

Verify that the OpenStack services and the rabbitmq have all re-started, and connected successfully by running the /opt/scm/scripts/OS_Status.sh script.

After all the openstack services report an **Up** status, start the mariadb container on the controller you just rebooted.

```
# docker start mariadb
```

Monitor the status of the database cluster as the database on the rebooted node synchronizes with the other two nodes in the cluster.

It may take several minutes for the synchronization to begin, so monitor the database for a few minutes before considering that the synchronization has started.

You need to proceed with caution here because when the the third instance connects with the two running instances the cluster size immediately jumps to 3 before galera discovers that the new instance needs to be updated.

The state during synchronization can be either Joining: receiving State Transfer or **Donor/Sync** depending on which mode of synchronization that galera decides to use

of either transferring the deltas or the entire database.

Variable_name	Value
wsrep_last_committed	3842876
wsrep_local_state_comment	Joining: receiving State Transfer
wsrep_incoming_addresses	10.10.3.1:3306,10.10.3.5:3 306,10.10.3.3:3306
wsrep_cluster_size	3
wsrep_cluster_status	Primary
wsrep_ready	OFF

After the synchronization completes the results will be similar to the following:

Variable_name	Value
wsrep_last_committed	3868364
wsrep_local_state_comment	Synced
wsrep_incoming_addresses	10.10.3.1:3306,10.10.3.5:3 306,10.10.3.3:3306
wsrep_cluster_size	3
wsrep_cluster_status	Primary
wsrep_ready	ON

Give the database some time to be safe and check that all the OpenStack services are up and running before starting on the next controller. This may be a good time for a coffee break. Check the status again when you return.

Carefully restarting the controllers one at a time will allow you to maintain the controllers without experiencing downtime.

1.18.5 Recovery from Failures

1.18.5.1 Network Failure or Power Loss

After a network failure, or a power loss check on the status of the database by running the `galeraStatus.sh`.

If the database did not recover, the report may resemble the following:

Variable_name	Value
wsrep_last_committed	3885809
wsrep_local_state_comment	Initialized
wsrep_incoming_addresses	10.10.3.5:3306
wsrep_cluster_size	1
wsrep_cluster_status	non-Primary
wsrep_ready	OFF

To recover, run the kolla ansible playbook to startup the database. The playbook will interrogate each instance of the database to find the latest committed transaction ID. The playbook will then bootstrap the database using that instance.

Before running the playbook, stop the database containers on all three controllers using the command:

```
# docker stop mariadb
```

From the ops system run the kolla ansible playbook to restart the database.

```
# kolla-ansible -i inventory/<clustername> mariadb_recovery
```

The mariadb_recovery playbook is not always successful but it does discover which controller has the most up to date version of the database. So make note of which controller the playbook chooses to start first as the playbook runs.

1.18.5.2 Worst Case - mariadb_recovery Failed

On the controller with the most up to date version of the database edit `/etc/kolla/mariadb/galera.cnf`.

Change the wsrep_cluster_address to the following value. Here we save the original value by commenting out the line.

Change:

```
wsrep_cluster_address = gcomm://10.11.0.1:4567,10.11.0.8:4567,10.11.0.9:4567
```

to:

```
#wsrep_cluster_address = gcomm://10.11.0.1:4567,10.11.0.8:4567,10.11.0.9:4567
wsrep_cluster_address = gcomm://
```

Then edit `/var/lib/docker/volumes/mariadb/_data/grastate.dat`.

Change the value of **safe_to_bootstrap** from 0 to 1.

```
safe_to_bootstrap: 0
```

to:

```
safe_to_bootstrap: 1
```

Next Start mariadb on this just this one controller:

```
# docker start mariadb
```

On this controller in a separate window monitor the mariadb startup log:

```
# tail -f /var/lib/docker/volumes/kolla_logs/_data/mariadb/mariadb.log
```

If there are no obvious errors in the log and the database starts running, cancel the monitoring using tail with a CTRL-C and change to the ops management window to monitor the database with the script, `galeraStatus.sh`.

Variable_name	Value
wsrep_last_committed	140856471
wsrep_local_state_comment	Synced
wsrep_incoming_addresses	10.11.0.9:3306
wsrep_cluster_size	1
wsrep_cluster_status	Primary
wsrep_ready	ON

Once you see the cluster state of **Synced**, size of **1**, status of **Primary** and wsrep_ready **ON**, start the mariadb container on the second controller.

Again monitor the startup and you will see the data sync over to this node:

Variable_name	Value
wsrep_last_committed	140856195
wsrep_local_state_comment	Joining: receiving State Transfer
wsrep_incoming_addresses	10.11.0.9:3306,10.11.0.8:3 306
wsrep_cluster_size	2
wsrep_cluster_status	Primary
wsrep_ready	OFF

Rerun the `galeraStatus.sh` script until you see the following:

Variable_name	Value
wsrep_last_committed	140857617
wsrep_local_state_comment	Synced
wsrep_incoming_addresses	10.11.0.9:3306,10.11.0.8:3 306
wsrep_cluster_size	2
wsrep_cluster_status	Primary
wsrep_ready	ON

Finally start mariadb on the last controller and monitor.

Variable_name	Value
wsrep_last_committed	140856155
wsrep_local_state_comment	Joining: receiving State Transfer
wsrep_incoming_addresses	10.11.0.1:3306,10.11.0.9:3 306,10.11.0.8:3306
wsrep_cluster_size	3
wsrep_cluster_status	Primary
wsrep_ready	OFF

You should see the state change to **Synced**, size of **3**, status of **Primary** and wsrep_ready of **ON**.

Variable_name	Value
wsrep_last_committed	140858943
wsrep_local_state_comment	Synced
wsrep_incoming_addresses	10.11.0.1:3306,10.11.0.9:3 306,10.11.0.8:3306
wsrep_cluster_size	3
wsrep_cluster_status	Primary
wsrep_ready	ON

The database is recovered!

To clean up the controller where you started, you will want to change the settings back to their original values.

On the controller where you started:

```
# docker stop mariadb
```

Again, edit the file `/etc/kolla/mariadb/galera.cnf`.

```
# docker start mariadb
```

and check `galeraStatus.sh` to verify the controller rejoins galera successfully.

1.18.6 When One Instance Will Not Start

In rare cases one of the database instances will not start and you will see errors in the `/var/lib/docker/volumes/kolla_logs/_data/mariadb/mariadb.log` logfile.

If the other 2 instances are working and are synced you can quickly recover this corrupted instance by letting galera run a full sync of the database to replace the corrupted data. This is done by first stopping the mariadb container that is stuck in a restarting state, then removing the databases and finally start the mariadb container again.

```
# docker stop mariadb
# rm -rf /var/lib/docker/volumes/mariadb/_data/
# docker start mariadb
```

Verify that the database transfer completed by listing the directory:

```
# ls -lh /var/lib/docker/volumes/mariadb/_data/
```

Also verify that the cluster is again healthy with the `galeraStatus.sh` script.

1.18.7 References

https://docs.openstack.org/kolla/newton/

https://docs.openstack.org/kolla-ansible/latest/

http://galeracluster.com/2016/11/introducing-the-safe-to-bootstrap-feature-in-galera-cluster

http://galeracluster.com/documentation-webpages/restartingcluster.html#safe-to-bootstrap-protection

http://galeracluster.com/documentation-webpages/galeraparameters.html#gcache-recover

1.19 Diagnostics and Troubleshooting

1.19.1 SCM Controller Services

All of the following services should be running on the SCM controller with the given ports, and writing to the given logs.

The Non HA Port is used by services in single-controller setups. In dual-controller setups, HAProxy listens on the service's normal (non HA) port and forwards to the service listening at the HA port.

You can check if a port is listening with Linux commands like:

```
# lsof -i :port
```

Service	Description	Non HA Port	HA Port	Log
Scyld Cloud Portal	SCM Web Dashboard	443	7690	/var/log/httpd/cloudportal/ /var/www/wsgi/cloudportal/cloudportal.log
Scyld Cloud Auth	SCM Authentication and Authorization	443	7691	/var/log/httpd/cloudauth/ /var/www/wsgi/cloudauth/cloudauth.log
Scyld Cloud Accountant	SCM Accounting	443	7692	/var/log/httpd/cloudaccountant/ /var/www/wsgi/cloudaccountant/cloudaccountant.log
Scyld Cloud Controller	SCM-OpenStack Interface	443	7693	/var/log/httpd/cloudcontroller/ /var/www/wsgi/cloudcontroller/cloudcontroller.log
Redis	In-memory database	6379		/var/log/redis/redis-server.log
Haproxy (if present)	HA proxy service	8080 (Stats)		/var/log/haproxy.log

1.19.2 OpenStack Services

OpenStack services are run inside Docker containers running on each OpenStack controller.

To check the health of the Docker containers on a controller:

```
# docker ps -a
[root@cc1-d ~]# docker ps
CONTAINER ID        IMAGE
↪ COMMAND             CREATED             STATUS              PORTS
↪NAMES
9e97dbdf37a9        k4e.scm:4000/kolla/centos-binary-grafana:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪grafana
3d0d128f4b32        k4e.scm:4000/kolla/centos-binary-gnocchi-statsd:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪gnocchi_statsd
16e1bbb9f94d        k4e.scm:4000/kolla/centos-binary-gnocchi-metricd:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪gnocchi_metricd
2f137db7e998        k4e.scm:4000/kolla/centos-binary-gnocchi-api:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪gnocchi_api
1fc44027a3eb        k4e.scm:4000/kolla/centos-binary-horizon:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪horizon
ee806258eeca        k4e.scm:4000/kolla/centos-binary-heat-engine:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪heat_engine
8c48feaa647a        k4e.scm:4000/kolla/centos-binary-heat-api-cfn:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪heat_api_cfn
b03f7cf87248        k4e.scm:4000/kolla/centos-binary-heat-api:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪heat_api
ee130fc0fbf9        k4e.scm:4000/kolla/centos-binary-cinder-backup:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪cinder_backup
1152a577e92a        k4e.scm:4000/kolla/centos-binary-cinder-volume:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪cinder_volume
```

(continues on next page)

(continued from previous page)

```
cf6a9c0e015d        k4e.scm:4000/kolla/centos-binary-cinder-scheduler:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪cinder_scheduler
10c43d513d5b        k4e.scm:4000/kolla/centos-binary-cinder-api:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪cinder_api
5c4a5573e3a2        k4e.scm:4000/kolla/centos-binary-neutron-metadata-agent:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪neutron_metadata_agent
a97881bd17f2        k4e.scm:4000/kolla/centos-binary-neutron-lbaas-agent:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪neutron_lbaas_agent
d106b0e253ca        k4e.scm:4000/kolla/centos-binary-neutron-l3-agent:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪neutron_l3_agent
7530545a141f        k4e.scm:4000/kolla/centos-binary-neutron-dhcp-agent:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪neutron_dhcp_agent
a8d52deb9ed2        k4e.scm:4000/kolla/centos-binary-neutron-openvswitch-agent:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪neutron_openvswitch_agent
0e3077090730        k4e.scm:4000/kolla/centos-binary-neutron-server:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪neutron_server
0f3007c497a1        k4e.scm:4000/kolla/centos-binary-nova-conductor:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪nova_conductor
9e5461eab29c        k4e.scm:4000/kolla/centos-binary-nova-spicehtml5proxy:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪nova_spicehtml5proxy
a7e1e000587d        k4e.scm:4000/kolla/centos-binary-nova-scheduler:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪nova_scheduler
fe20eb20c8d5        k4e.scm:4000/kolla/centos-binary-nova-consoleauth:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪nova_consoleauth
da68b9764cd4        k4e.scm:4000/kolla/centos-binary-nova-api:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪nova_api
aac9176c907e        k4e.scm:4000/kolla/centos-binary-glance-api:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪glance_api
0a096c67003e        k4e.scm:4000/kolla/centos-binary-glance-registry:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪glance_registry
37202d5529f7        k4e.scm:4000/kolla/centos-binary-keystone:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪keystone
11dd5af66a86        k4e.scm:4000/kolla/centos-binary-rabbitmq:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪rabbitmq
7e65e2a8825b        k4e.scm:4000/kolla/centos-binary-mariadb:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪mariadb
cca6cdc74245        k4e.scm:4000/kolla/centos-binary-memcached:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪memcached
```

(continues on next page)

(continued from previous page)

```
796247e2a783        k4e.scm:4000/kolla/centos-binary-kibana:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪kibana
e0d6f0a0c25e        k4e.scm:4000/kolla/centos-binary-keepalived:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪keepalived
9abb6ca64f2d        k4e.scm:4000/kolla/centos-binary-haproxy:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪haproxy
cb4f91e02914        k4e.scm:4000/kolla/centos-binary-elasticsearch:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪elasticsearch
4cf2280d2854        k4e.scm:4000/kolla/centos-binary-ceph-rgw:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪ceph_rgw
c34e2b58edba        k4e.scm:4000/kolla/centos-binary-ceph-osd:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪ceph_osd_0
c0349b4d3650        k4e.scm:4000/kolla/centos-binary-ceph-mon:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪ceph_mon
afa384731c7b        k4e.scm:4000/kolla/centos-binary-cron:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪cron
e3003c071f19        k4e.scm:4000/kolla/centos-binary-kolla-toolbox:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪kolla_toolbox
c7eee2e64766        k4e.scm:4000/kolla/centos-binary-heka:3.0.2
↪ "kolla_start"       2 weeks ago         Up 2 weeks
↪heka
```

Log files for each service are available from Docker by running `docker logs SERVICE_NAME` or under `/var/lib/docker/volumes/kolla_logs/_data/SERVICE_NAME`.

A web dashboard for logs, Kibana, is also available at the same hostname that provides the Horizon dashboard at port 5601.

1.19.3 Ceph

To get a quick Ceph status check, run `ceph health` on a controller. You should get this response:

```
# ceph health
HEALTH_OK
```

If the Ceph `noout` option has been set, you'll get this response instead:

```
# ceph health
HEALTH_WARN noout flag(s) set
```

More details are available with `ceph -s`:

```
# ceph -s
    cluster 8c213314-82f6-44ea-bcdc-f8d44a93186b
     health HEALTH_WARN
            noout flag(s) set
```

(continues on next page)

(continued from previous page)

```
    monmap e1: 4 mons at {cc1-d=10.10.4.1:6789/0,vmhost1-d=10.10.4.3:6789/0,vmhost2-
↪d=10.10.4.4:6789/0,vmhost3-d=10.10.4.5:6789/0}
           election epoch 76, quorum 0,1,2,3 cc1-d,vmhost1-d,vmhost2-d,vmhost3-d
    osdmap e42: 4 osds: 4 up, 4 in
           flags noout
     pgmap v19117: 384 pgs, 3 pools, 13272 MB data, 1674 objects
           26745 MB used, 3678 GB / 3704 GB avail
                384 active+clean
  client io 11374 kB/s rd, 9661 B/s wr, 188 op/s
```

1.20 Glossary

API Application programming Interface, specifies how some software components should interact with each other.

Ceph A free software storage platform designed to present object, block, and file storage from a single distributed cluster.

Cinder Cinder is a Block Storage service for OpenStack.

CLI Command-line interface is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

COW Copy-on-write is an optimization strategy used in computer programming.

DHCP Dynamic Host Configuration Protocol is a standardized networking protocol used on Internet Protocol (IP) networks for dynamically distributing network configuration parameters, such as IP addresses for interfaces and services.

FOSS Free, open-source software.

Galera Galera Cluster is an easy-to-use, high-availability solution, which provides high system uptime, no data loss, and scalability for future growth.

GbE Gigabiits per second Ethernet.

Glance The OpenStack Glance project provides services for discovering, registering, and retrieving virtual machine images.

HAProxy A free, fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications by spreading requests across multiple servers.

HPC High Performance Computing most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business.

HTTPS Hypertext Transfer Protocol Secure is a communications protocol for secure communication over a computer network, with especially wide deployment on the Internet.

IaaS Infrastructure as a Service is a provision model in which an organization outsources the equipment used to support operations, including storage, hardware, servers and networking components.

Instance Type A server-instance defines the resources for a virtual machine, meaning its RAM, core count, hard disk space.

JSON JavaScript Object Notation is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.

keepalived A routing software written in C, that provides simple but robust facilities for load balancing servers and a high-availability to Linux system and Linux based infrastructures.

LACP Link aggregation control protocol, describe various methods of combining (aggregating) multiple network connections in parallel to increase throughput beyond what a single connection could sustain, and to provide redundancy in case one of the links fail.

LDAP Lightweight Directory Access Protocol is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.

MLAG The ability of two and sometimes more switches to act like a single switch when forming link bundles.

MySQL An open-source relational database management system (RDBMS).

NAT Network address translation is a methodology of modifying network address information in Internet Protocol (IP) datagram packet headers while they are in transit across a traffic routing device for the purpose of remapping one IP address space into another.

NIS The Network Information Service (originally called Yellow Pages or YP) is a client–server directory service protocol for distributing system configuration data such as user and host names between computers on a computer network.

OAuth An open standard for authorization. OAuth provides client applications a 'secure delegated access' to server resources on behalf of a resource owner.

OpenStack A free and open-source software cloud computing platform, primarily deployed as an infrastructure-as-a-solution (IaaS) and released under the terms of the Apache License.

POD Penguin Computing On Demand.

Python A widely used general-purpose, high-level programming language.

RAM Random-access memory is a form of computer data storage.

RBAC Role-based Access Control is an approach to restricting system access to authorized users.

SCM Scyld Cloud Manager.

server-instance Synonymous with Virtual Machine. It is a virtualized compute resource comprised of an instance type and an image.

SGE Sun Grid Engine is a grid computing computer cluster software system (otherwise known as batch-queuing system).

SMTP Simple Mail Transfer Protocol is an Internet standard for email transmission.

SSH A cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers.

UGE Univa Grid Engine is a batch-queuing system, forked from Sun Grid Engine (SGE).

UUID A universally unique identifier is an identifier standard used in software construction, standardized by the Open Software Foundation as part of the Distributed Computing Environment.

VIP Virtual Internet Protocol address.