nova-powervm Documentation

Release 3.0.4.dev3

IBM

July 13, 2018

Contents

This project provides a Nova-compatible compute driver for PowerVM systems.

The project aims to integrate into OpenStack's Nova project. Initial development is occurring in a separate project until it has matured and met the Nova core team's requirements. As such, all development practices should mirror those of the Nova project.

Documentation on Nova can be found at the Nova Devref.

CHAPTER 1

Nova-PowerVM Overview

Contents:

PowerVM Nova Driver

The IBM PowerVM hypervisor provides virtualization on POWER hardware. PowerVM admins can see benefits in their environments by making use of OpenStack. This driver (along with a Neutron ML2 compatible agent and Ceilometer agent) provides the capability for operators of PowerVM to use OpenStack natively.

Problem Description

As ecosystems continue to evolve around the POWER platform, a single OpenStack driver does not meet all of the needs for the various hypervisors. The standard libvirt driver provides support for KVM on POWER systems. This nova driver provides PowerVM support to OpenStack environment.

This driver meets the following:

- Built within the community
- Fits the OpenStack model
- Utilizes automated functional and unit tests
- Enables use of PowerVM systems through the OpenStack APIs
- Allows attachment of volumes from Cinder over supported protocols

This driver makes the following use cases available for PowerVM:

- As a deployer, all of the standard lifecycle operations (start, stop, reboot, migrate, destroy, etc.) should be supported on a PowerVM based instance.
- As a deployer, I should be able to capture an instance to an image.
- VNC console to instances deployed.

Usage

To use the driver, install the nova-powervm project on your NovaLink-based PowerVM system. The nova-powervm project has a minimal set of configuration. See the configuration options section of the dev-ref for more information.

It is recommended that operators also make use of the networking-powervm project. The project ensures that the network bridge supports the VLAN-based networks required for the workloads.

There is also a ceilometer-powervm project that can be included.

Future work will be done to include PowerVM into the various OpenStack deployment models.

Overview of Architecture

The driver enables the following:

- Provide deployments that work with the OpenStack model.
- Driver is implemented using a new version of the PowerVM REST API.
- Ephemeral disks are supported either with Virtual I/O Server (VIOS) hosted local disks or via Shared Storage Pools (a PowerVM cluster file system).
- Volume support is provided via Cinder through supported protocols for the Hypervisor (virtual SCSI and N-Port ID Virtualization).
- Live migration support is available when using Shared Storage Pools or boot from volume.
- Network integration is supported via the ML2 compatible Neutron Agent. This is the openstack/networking-powervm project.
- Automated Functional Testing is provided to validate changes from the broader OpenStack community against the PowerVM driver.
- Thorough unit, syntax, and style testing is provided and enforced for the driver.

The intention is that this driver follows the OpenStack Nova model and will be a candidate for promotion (via a subsequent blueprint) into the nova core project.

Data Model Impact

- The evacuate API is supported as part of the PowerVM driver. It optionally allows for the NVRAM data to be stored to a Swift database. However this does not impact the data model itself. It simply provides a location to optionally store the VM's NVRAM metadata in the event of a rebuild, evacuate, shelve, migration or resize.

REST API Impact

No REST API impacts.

Security Impact

No known security impacts.

Notifications Impact

No new notifications. The driver does expect that the Neutron agent will return an event when the VIF plug has occurred, assuming that Neutron is the network service.

Other End User Impact

The administrator may notice new logging messages in the nova compute logs.

Performance Impact

The driver has a similar deployment speed and agility to other hypervisors. It has been tested with up to 10 concurrent deploys with several hundred VMs on a given server.

Most operations are comparable in speed. Deployment, attach/detach volumes, lifecycle, etc... are quick.

Due to the nature of the project, any performance impacts are limited to the Compute Driver. The API processes for instance are not impacted.

Other Deployer Impact

The cloud administrator will need to refer to documentation on how to configure OpenStack for use with a PowerVM hypervisor.

A 'powervm' configuration group is used to contain all the PowerVM specific configuration settings. Existing configuration file attributes will be reused as much as possible (e.g. vif_plugging_timeout). This reduces the number of PowerVM specific items that will be needed.

It is the goal of the project to only require minimal additional attributes. The deployer may specify additional attributes to fit their configuration.

Live Migration

In the Mitaka release, the Nova project moved to using conductor-based objects for the live migration flow. These objects exist in nova/objects/migrate_data.py.

While the PowerVM driver supports live migration, it can not register its own live migration object due to being out of tree. The team is working with the nova core team to bring the PowerVM driver in tree. However until that time, the use of live migration with the PowerVM driver requires starting a PowerVM-specific conductor.

This conductor does not limit the OpenStack cloud to only supporting PowerVM. Rather, it simply allows an existing cloud to include PowerVM support within it.

To use the conductor, install the nova-powervm project on the node running the nova conductor. Then start the 'nova-conductor-powervm' process. This will support ALL of the hypervisors, including PowerVM.

To reiterate, this is only needed if you plan to use PowerVM's live migrate functionality.

Developer Impact

The code for this driver is currently contained within a powervm project. The driver is within the /nova_powervm/virt/powervm/ package and extends the nova.virt.driver.ComputeDriver class.

The code interacts with PowerVM through the pypowervm library. This python binding is a wrapper to the PowerVM REST API. All hypervisor operations interact with the PowerVM REST API via this binding. The driver is maintained to support future revisions of the PowerVM REST API as needed.

For ephemeral disk support, either a Virtual I/O Server hosted local disk or a Shared Storage Pool (a PowerVM clustered file system) is supported. For volume attachments, the driver supports Cinder-based attachments via protocols supported by the hypervisor (e.g. Fibre Channel).

For networking, the networking-powervm project provides Neutron ML2 Agents. The agents provide the necessary configuration on the Virtual I/O Server for networking. The PowerVM Nova driver code creates the VIF for the client VM, but the Neutron agent creates the VIF for VLANs.

Automated functional testing is provided through a third party continuous integration system. It monitors for incoming Nova change sets, runs a set of functional tests (lifecycle operations) against the incoming change, and provides a non-gating vote (+1 or -1).

Developers should not be impacted by these changes unless they wish to try the driver.

Community Impact

The intent of this project is to bring another driver to OpenStack that aligns with the ideals and vision of the community. The intention is to promote this to core Nova.

Alternatives

No alternatives appear viable to bring PowerVM support into the OpenStack community.

Implementation

Assignee(s)

Primary assignees: adreznec efried kyleh thorst

Other contributors: multiple

Dependencies

- Utilizes the PowerVM REST API specification for management. Will utilize future versions of this specification as it becomes available: http://ibm.co/1lThV9R
- Builds on top of the pypowervm library. This is a prerequisite to utilizing the driver.

Testing

Tempest Tests

Since the tempest tests should be implementation agnostic, the existing tempest tests should be able to run against the PowerVM driver without issue.

Tempest tests that require function that the platform does not yet support (e.g. iSCSI or Floating IPs) will not pass. These should be ommitted from the Tempest test suite.

A *sample Tempest test configuration* for the PowerVM driver has been provided.

Thorough unit tests exist within the project to validate specific functions within this implementation.

Functional Tests

A third party functional test environment has been created. It monitors for incoming nova change sets. Once it detects a new change set, it will execute the existing lifecycle API tests. A non-gating vote (+1 or -1) will be provided with information provided (logs) based on the result.

API Tests

Existing APIs should be valid. All testing is planned within the functional testing system and via unit tests.

Documentation Impact

User Documentation

See the dev-ref for documentation on how to configure, contribute, use, etc. this driver implementation.

Developer Documentation

The existing Nova developer documentation should typically suffice. However, until merge into Nova, we will maintain a subset of dev-ref documentation.

References

- PowerVM REST API Specification (may require newer versions as they become available): http://ibm.co/1lThV9R
- PowerVM Virtualization Introduction and Configuration: http://www.redbooks.ibm.com/abstracts/sg247940.html
- PowerVM Best Practices: http://www.redbooks.ibm.com/abstracts/sg248062.html

Feature Support Matrix

> **Warning:** Please note, while this document is still being maintained, this is slowly being updated to re-group and classify features

When considering which capabilities should be marked as mandatory the following general guiding principles were applied

- **Inclusivity** - people have shown ability to make effective use of a wide range of virtualization technologies with broadly varying featuresets. Aiming to keep the requirements as inclusive as possible, avoids second-guessing what a user may wish to use the cloud compute service for.
- **Bootstrapping** - a practical use case test is to consider that starting point for the compute deploy is an empty data center with new machines and network connectivity. The look at what are the minimum features required of a compute service, in order to get user instances running and processing work over the network.
- **Competition** - an early leader in the cloud compute service space was Amazon EC2. A sanity check for whether a feature should be mandatory is to consider whether it was available in the first public release of EC2. This had quite a narrow featureset, but none the less found very high usage in many use cases. So it serves to illustrate that many features need not be considered mandatory in order to get useful work done.
- **Reality** - there are many virt drivers currently shipped with Nova, each with their own supported feature set. Any feature which is missing in at least one virt driver that is already in-tree, must by inference be considered optional until all in-tree drivers support it. This does not rule out the possibility of a currently optional feature becoming mandatory at a later date, based on other principles above.

Summary

Feature	*Status*	**PowerVM**
Attach block volume to instance	optional	
Detach block volume from instance	optional	
Set the host in a maintenance mode	optional	
Evacuate instances from a host	optional	
Rebuild instance	optional	
Guest instance status	mandatory	
Guest host status	optional	
Live migrate instance across hosts	optional	
Launch instance	mandatory	
Stop instance CPUs (pause)	optional	
Reboot instance	optional	
Rescue instance	optional	
Resize instance	optional	
Restore instance	optional	
Service control	optional	
Set instance admin password	optional	
Save snapshot of instance disk	optional	
Suspend instance	optional	
Swap block volumes	optional	
Shutdown instance	mandatory	
Trigger crash dump	optional	
Resume instance CPUs (unpause)	optional	
Auto configure disk	optional	
Instance disk I/O limits	optional	
Config drive support	choice	
Inject files into disk image	optional	
Inject guest networking config	optional	
Remote desktop over RDP	choice	
View serial console logs	choice	
Remote interactive serial console	choice	
Remote desktop over SPICE	choice	
Remote desktop over VNC	choice	
Block storage support	optional	
Block storage over fibre channel	optional	
Block storage over iSCSI	condition	
CHAP authentication for iSCSI	optional	
Image storage support	mandatory	
Network firewall rules	optional	
Network routing	optional	
Network security groups	optional	
Flat networking	choice	
VLAN networking	choice	
uefi boot	optional	

Details

- **Attach block volume to instance Status: optional.** The attach volume operation provides a means to hotplug additional block storage to a running instance. This allows storage capabilities to be expanded without interruption of service. In a cloud model it would be more typical to just spin up a new instance with large storage, so the ability to hotplug extra storage is for those cases where the instance is considered to be more of a pet than cattle. Therefore this operation is not considered to be mandatory to support.

  drivers:

  - **PowerVM:** `complete`

- **Detach block volume from instance Status: optional.** See notes for attach volume operation.

  drivers:

  - **PowerVM:** `complete`

- **Set the host in a maintenance mode Status: optional.** This operation allows a host to be placed into maintenance mode, automatically triggering migration of any running instances to an alternative host and preventing new instances from being launched. This is not considered to be a mandatory operation to support. The CLI command is "nova host-update <host>". The driver methods to implement are "host_maintenance_mode" and "set_host_enabled".

  drivers:

  - **PowerVM:** `complete`

- **Evacuate instances from a host Status: optional.** A possible failure scenario in a cloud environment is the outage of one of the compute nodes. In such a case the instances of the down host can be evacuated to another host. It is assumed that the old host is unlikely ever to be powered back on, otherwise the evacuation attempt will be rejected. When the instances get moved to the new host, their volumes get re-attached and the locally stored data is dropped. That happens in the same way as a rebuild. This is not considered to be a mandatory operation to support.

  CLI commands:

  - `nova evacuate <server>`
  - `nova host-evacuate <host>`

  drivers:

  - **PowerVM:** `complete`

- **Rebuild instance Status: optional.** A possible use case is additional attributes need to be set to the instance, nova will purge all existing data from the system and remakes the VM with given information such as 'metadata' and 'personalities'. Though this is not considered to be a mandatory operation to support.

  CLI commands:

  - `nova rebuild <server> <image>`
  -

  drivers:

  - **PowerVM:** `complete`

- **Guest instance status Status: mandatory.** Provides a quick report on information about the guest instance, including the power state, memory allocation, CPU allocation, number of vCPUs and cummulative CPU execution time. As well as being informational, the power state is used by the compute manager for tracking changes in guests. Therefore this operation is considered mandatory to support.

  drivers:

  - **PowerVM:** `complete`

- **Guest host status Status: optional.** Unclear what this refers to

  drivers:

  - **PowerVM:** `complete`

- **Live migrate instance across hosts Status: optional.** Live migration provides a way to move an instance off one compute host, to another compute host. Administrators may use this to evacuate instances from a host that

needs to undergo maintenance tasks, though of course this may not help if the host is already suffering a failure. In general instances are considered cattle rather than pets, so it is expected that an instance is liable to be killed if host maintenance is required. It is technically challenging for some hypervisors to provide support for the live migration operation, particularly those built on the container based virtualization. Therefore this operation is not considered mandatory to support.

drivers:

- **PowerVM:** `complete`

- **Launch instance Status: mandatory.** Importing pre-existing running virtual machines on a host is considered out of scope of the cloud paradigm. Therefore this operation is mandatory to support in drivers.

  drivers:

  - **PowerVM:** `complete`

- **Stop instance CPUs (pause) Status: optional.** Stopping an instances CPUs can be thought of as roughly equivalent to suspend-to-RAM. The instance is still present in memory, but execution has stopped. The problem, however, is that there is no mechanism to inform the guest OS that this takes place, so upon unpausing, its clocks will no longer report correct time. For this reason hypervisor vendors generally discourage use of this feature and some do not even implement it. Therefore this operation is considered optional to support in drivers.

  drivers:

  - **PowerVM:** `missing`

- **Reboot instance Status: optional.** It is reasonable for a guest OS administrator to trigger a graceful reboot from inside the instance. A host initiated graceful reboot requires guest co-operation and a non-graceful reboot can be achieved by a combination of stop+start. Therefore this operation is considered optional.

  drivers:

  - **PowerVM:** `complete`

- **Rescue instance Status: optional.** The rescue operation starts an instance in a special configuration whereby it is booted from an special root disk image. The goal is to allow an administrator to recover the state of a broken virtual machine. In general the cloud model considers instances to be cattle, so if an instance breaks the general expectation is that it be thrown away and a new instance created. Therefore this operation is considered optional to support in drivers.

  drivers:

  - **PowerVM:** `complete`

- **Resize instance Status: optional.** The resize operation allows the user to change a running instance to match the size of a different flavor from the one it was initially launched with. There are many different flavor attributes that potentially need to be updated. In general it is technically challenging for a hypervisor to support the alteration of all relevant config settings for a running instance. Therefore this operation is considered optional to support in drivers.

  drivers:

  - **PowerVM:** `complete`

- **Restore instance Status: optional.** See notes for the suspend operation

  drivers:

  - **PowerVM:** `missing`

- **Service control Status: optional.** Something something, dark side, something something. Hard to claim this is mandatory when no one seems to know what "Service control" refers to in the context of virt drivers.

  drivers:

  - **PowerVM:** `missing`

- **Set instance admin password Status: optional.** Provides a mechanism to re(set) the password of the administrator account inside the instance operating system. This requires that the hypervisor has a way to communicate with the running guest operating system. Given the wide range of operating systems in existence it is unreasonable to expect this to be practical in the general case. The configdrive and metadata service both provide a mechanism for setting the administrator password at initial boot time. In the case where this operation were not available, the administrator would simply have to login to the guest and change the password in the normal manner, so this is just a convenient optimization. Therefore this operation is not considered mandatory for drivers to support.

  drivers:

  - **PowerVM:** `missing`

- **Save snapshot of instance disk Status: optional.** The snapshot operation allows the current state of the instance root disk to be saved and uploaded back into the glance image repository. The instance can later be booted again using this saved image. This is in effect making the ephemeral instance root disk into a semi-persistent storage, in so much as it is preserved even though the guest is no longer running. In general though, the expectation is that the root disks are ephemeral so the ability to take a snapshot cannot be assumed. Therefore this operation is not considered mandatory to support.

  drivers:

  - **PowerVM:** `complete`

- **Suspend instance Status: optional.** Suspending an instance can be thought of as roughly equivalent to suspend-to-disk. The instance no longer consumes any RAM or CPUs, with its live running state having been preserved in a file on disk. It can later be restored, at which point it should continue execution where it left off. As with stopping instance CPUs, it suffers from the fact that the guest OS will typically be left with a clock that is no longer telling correct time. For container based virtualization solutions, this operation is particularly technically challenging to implement and is an area of active research. This operation tends to make more sense when thinking of instances as pets, rather than cattle, since with cattle it would be simpler to just terminate the instance instead of suspending. Therefore this operation is considered optional to support.

  drivers:

  - **PowerVM:** `missing`

- **Swap block volumes Status: optional.** The swap volume operation is a mechanism for changing a running instance so that its attached volume(s) are backed by different storage in the host. An alternative to this would be to simply terminate the existing instance and spawn a new instance with the new storage. In other words this operation is primarily targeted towards the pet use case rather than cattle, however, it is required for volume migration to work in the volume service. This is considered optional to support.

  drivers:

  - **PowerVM:** `missing`

- **Shutdown instance Status: mandatory.** The ability to terminate a virtual machine is required in order for a cloud user to stop utilizing resources and thus avoid indefinitely ongoing billing. Therefore this operation is mandatory to support in drivers.

  drivers:

  - **PowerVM:** `complete`

- **Trigger crash dump Status: optional.** The trigger crash dump operation is a mechanism for triggering a crash dump in an instance. The feature is typically implemented by injecting an NMI (Non-maskable Interrupt) into the instance. It provides a means to dump the production memory image as a dump file which is useful for users. Therefore this operation is considered optional to support.

drivers:

- **PowerVM:** `missing`

- **Resume instance CPUs (unpause) Status: optional.** See notes for the "Stop instance CPUs" operation

  drivers:

  - **PowerVM:** `missing`

- **Auto configure disk Status: optional.** something something, dark side, something something. Unclear just what this is about.

  drivers:

  - **PowerVM:** `missing`

- **Instance disk I/O limits Status: optional.** The ability to set rate limits on virtual disks allows for greater performance isolation between instances running on the same host storage. It is valid to delegate scheduling of I/O operations to the hypervisor with its default settings, instead of doing fine grained tuning. Therefore this is not considered to be an mandatory configuration to support.

  drivers:

  - **PowerVM:** `missing`

- **Config drive support Status: choice(guest.setup).** The config drive provides an information channel into the guest operating system, to enable configuration of the administrator password, file injection, registration of SSH keys, etc. Since cloud images typically ship with all login methods locked, a mechanism to set the administrator password of keys is required to get login access. Alternatives include the metadata service and disk injection. At least one of the guest setup mechanisms is required to be supported by drivers, in order to enable login access.

  drivers:

  - **PowerVM:** `complete`

- **Inject files into disk image Status: optional.** This allows for the end user to provide data for multiple files to be injected into the root filesystem before an instance is booted. This requires that the compute node understand the format of the filesystem and any partitioning scheme it might use on the block device. This is a non-trivial problem considering the vast number of filesystems in existence. The problem of injecting files to a guest OS is better solved by obtaining via the metadata service or config drive. Therefore this operation is considered optional to support.

  drivers:

  - **PowerVM:** `missing`

- **Inject guest networking config Status: optional.** This allows for static networking configuration (IP address, netmask, gateway and routes) to be injected directly into the root filesystem before an instance is booted. This requires that the compute node understand how networking is configured in the guest OS which is a non-trivial problem considering the vast number of operating system types. The problem of configuring networking is better solved by DHCP or by obtaining static config via the metadata service or config drive. Therefore this operation is considered optional to support.

  drivers:

  - **PowerVM:** `missing`

- **Remote desktop over RDP Status: choice(console).** This allows the administrator to interact with the graphical console of the guest OS via RDP. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Some operating systems may prefer to emit messages via the serial console for easier consumption. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

drivers:

- **PowerVM:** `missing`

- **View serial console logs Status: choice(console).** This allows the administrator to query the logs of data emitted by the guest OS on its virtualized serial port. For UNIX guests this typically includes all boot up messages and so is useful for diagnosing problems when an instance fails to successfully boot. Not all guest operating systems will be able to emit boot information on a serial console, others may only support graphical consoles. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **PowerVM:** `missing`

- **Remote interactive serial console Status: choice(console).** This allows the administrator to interact with the serial console of the guest OS. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Not all guest operating systems will be able to emit boot information on a serial console, others may only support graphical consoles. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations. This feature was introduced in the Juno release with blueprint https://blueprints.launchpad.net/nova/+spec/serial-ports

  CLI commands:

  - `nova get-serial-console <server>`

  drivers:

  - **PowerVM:** `missing`

- **Remote desktop over SPICE Status: choice(console).** This allows the administrator to interact with the graphical console of the guest OS via SPICE. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Some operating systems may prefer to emit messages via the serial console for easier consumption. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **PowerVM:** `missing`

- **Remote desktop over VNC Status: choice(console).** This allows the administrator to interact with the graphical console of the guest OS via VNC. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Some operating systems may prefer to emit messages via the serial console for easier consumption. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **PowerVM:** `complete`

- **Block storage support Status: optional.** Block storage provides instances with direct attached virtual disks that can be used for persistent storage of data. As an alternative to direct attached disks, an instance may choose to use network based persistent storage. OpenStack provides object storage via the Swift service, or a traditional filesystem such as NFS/GlusterFS may be used. Some types of instances may not require persistent storage at all, being simple transaction processing systems reading requests & sending results to and from the network. Therefore support for this configuration is not considered mandatory for drivers to support.

  drivers:

  - **PowerVM:** `complete`

- **Block storage over fibre channel Status: optional.** To maximise performance of the block storage, it may be desirable to directly access fibre channel LUNs from the underlying storage technology on the compute hosts. Since this is just a performance optimization of the I/O path it is not considered mandatory to support.

  drivers:

  - **PowerVM:** `complete`

- **Block storage over iSCSI Status: condition(storage.block==complete).** If the driver wishes to support block storage, it is common to provide an iSCSI based backend to access the storage from cinder. This isolates the compute layer for knowledge of the specific storage technology used by Cinder, albeit at a potential performance cost due to the longer I/O path involved. If the driver chooses to support block storage, then this is considered mandatory to support, otherwise it is considered optional.

  drivers:

  - **PowerVM:** `missing`

- **CHAP authentication for iSCSI Status: optional.** If accessing the cinder iSCSI service over an untrusted LAN it is desirable to be able to enable authentication for the iSCSI protocol. CHAP is the commonly used authentication protocol for iSCSI. This is not considered mandatory to support. (?)

  drivers:

  - **PowerVM:** `missing`

- **Image storage support Status: mandatory.** This refers to the ability to boot an instance from an image stored in the glance image repository. Without this feature it would not be possible to bootstrap from a clean environment, since there would be no way to get block volumes populated and reliance on external PXE servers is out of scope. Therefore this is considered a mandatory storage feature to support.

  drivers:

  - **PowerVM:** `complete`

- **Network firewall rules Status: optional.** Unclear how this is different from security groups

  drivers:

  - **PowerVM:** `missing`

- **Network routing Status: optional.** Unclear what this refers to

  drivers:

  - **PowerVM:** `complete`

- **Network security groups Status: optional.** The security groups feature provides a way to define rules to isolate the network traffic of different instances running on a compute host. This would prevent actions such as MAC and IP address spoofing, or the ability to setup rogue DHCP servers. In a private cloud environment this may be considered to be a superfluous requirement. Therefore this is considered to be an optional configuration to support.

  drivers:

  - **PowerVM:** `missing`

- **Flat networking Status: choice(networking.topology).** Provide network connectivity to guests using a flat topology across all compute nodes. At least one of the networking configurations is mandatory to support in the drivers.

  drivers:

  - **PowerVM:** `complete`

- **VLAN networking Status: choice(networking.topology).** Provide network connectivity to guests using VLANs to define the topology. At least one of the networking configurations is mandatory to support in the drivers.

  drivers:

  - **PowerVM:** `complete`

- **uefi boot Status: optional.** This allows users to boot a guest with uefi firmware.

  drivers:

  - **PowerVM:** `missing`

Notes

- **Virtuozzo was formerly named Parallels in this document**

CHAPTER 2

Nova-PowerVM Policies

Contents:

Nova-PowerVM Policies

In the Policies Guide, you will find documented policies for developing with Nova-PowerVM. This includes the processes we use for blueprints and specs, bugs, contributor onboarding, and other procedural items.

Policies

Nova-PowerVM Bugs

Nova-PowerVM maintains all of its bugs in Launchpad. All of the current open Nova-PowerVM bugs can be found in that link.

Bug Triage Process

The process of bug triaging consists of the following steps:

1. Check if a bug was filed for a correct component (project). If not, either change the project or mark it as "Invalid".
2. Add appropriate tags. Even if the bug is not valid or is a duplicate of another one, it still may help bug submitters and corresponding sub-teams.
3. Check if a similar bug was filed before. If so, mark it as a duplicate of the previous bug.
4. Check if the bug description is consistent, e.g. it has enough information for developers to reproduce it. If it's not consistent, ask submitter to provide more info and mark a bug as "Incomplete".
5. Depending on ease of reproduction (or if the issue can be spotted in the code), mark it as "Confirmed".
6. Assign the importance. Bugs that obviously break core and widely used functionality should get assigned as "High" or "Critical" importance. The same applies to bugs that were filed for gate failures.
7. (Optional). Add comments explaining the issue and possible strategy of fixing/working around the bug.

Contributing to Nova-PowerVM

If you would like to contribute to the development of OpenStack, you must follow the steps in the "If you're a developer, start here" section of this page:

http://wiki.openstack.org/HowToContribute

Once those steps have been completed, changes to OpenStack should be submitted for review via the Gerrit tool, following the workflow documented at:

http://wiki.openstack.org/GerritWorkflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub:

https://bugs.launchpad.net/nova-powervm

Code Reviews

Code reviews are a critical component of all OpenStack projects. Code reviews provide a way to enforce a level of consistency across the project, and also allow for the careful onboarding of contributions from new contributors.

Code Review Practices

Nova-PowerVM follows the code review guidelines as set forth for all OpenStack projects. It is expected that all reviewers are following the guidelines set forth on that page.

Indices and tables

- *genindex*
- *modindex*
- *search*

CHAPTER 3

Nova-PowerVM Devref

Contents:

Developer Guide

In the Developer Guide, you will find information on how to develop for Nova-PowerVM and how it interacts with Nova compute. You will also find information on setup and usage of Nova-PowerVM

Internals and Programming

Source Code Structure

Since nova-powervm strives to be integrated into the upstream Nova project, the source code structure matches a standard driver.

```
nova_powervm/
  virt/
    powervm/
      disk/
      tasks/
      volume/
      ...
  tests/
    virt/
      powervm/
        disk/
        tasks/
        volume/
        ...
```

nova_powervm/virt/powervm

The main directory for the overall driver. Provides the driver implementation, image support, and some high level classes to interact with the PowerVM system (ex. host, vios, vm, etc...)

The driver attempts to utilize TaskFlow for major actions such as spawn. This allows the driver to create atomic elements (within the tasks) to drive operations against the system (with revert capabilities).

nova_powervm/virt/powervm/disk

The disk folder contains the various 'nova ephemeral' disk implementations. These are basic images that do not involve Cinder.

Two disk implementations exist currently.

- localdisk - supports Virtual I/O Server Volume Groups. This configuration uses any Volume Group on the system, allowing operators to make use of the physical disks local to their system.
- Shared Storage Pool - utilizes PowerVM's distributed storage. As such this implementation allows operators to make use of live migration capabilities.

The standard interface between these two implementations is defined in the driver.py. This ensures that the nova-powervm compute driver does not need to know the specifics about which disk implementation it is using.

nova_powervm/virt/powervm/tasks

The task folder contains TaskFlow classes. These implementations simply wrap around other methods, providing logical units that the compute driver can use when building a string of actions.

For instance, spawning an instance may require several atomic tasks:

- Create VM
- Plug Networking
- Create Disk from Glance
- Attach Disk to VM
- Power On

The tasks in this directory encapsulate this. If anything fails, they have corresponding reverts. The logic to perform these operations is contained elsewhere; these are simple wrappers that enable embedding into Taskflow.

nova_powervm/virt/powervm/volume

The volume folder contains the Cinder volume connectors. A volume connector is the code that connects a Cinder volume (which is visible to the host) to the Virtual Machine.

The PowerVM Compute Driver has an interface for the volume connectors defined in this folder's *driver.py*.

The PowerVM Compute Driver provides two implementations for Fibre Channel attached disks.

- Virtual SCSI (vSCSI): The disk is presented to a Virtual I/O Server and the data is passed through to the VM through a virtualized SCSI connection.
- N-Port ID Virtualization (NPIV): The disk is presented directly to the VM. The VM will have virtual Fibre Channel connections to the disk, and the Virtual I/O Server will not have the disk visible to it.

Setting Up a Development Environment

This page describes how to setup a working Python development environment that can be used in developing Nova-PowerVM.

These instructions assume you're already familiar with Git and Gerrit, which is a code repository mirror and code review toolset, however if you aren't please see this Git tutorial for an introduction to using Git and this guide for a tutorial on using Gerrit and Git for code contribution to OpenStack projects.

Getting the code

Grab the code:

```
git clone git://git.openstack.org/openstack/nova-powervm
cd nova-powervm
```

Setting up your environment

The purpose of this project is to provide the 'glue' between OpenStack Compute (Nova) and PowerVM. The pypowervm project is used to control PowerVM systems.

It is recommended that you clone down the OpenStack Nova project along with pypowervm into your respective development environment.

Running the tox python targets for tests will automatically clone these down via the requirements.

Additional project requirements may be found in the requirements.txt file.

Usage

To make use of the PowerVM drivers, a PowerVM system set up with NovaLink is required. The nova-powervm driver should be installed on the management VM.

Note: Installing the NovaLink software creates the `pvm_admin` group. In order to function properly, the user executing the Nova compute service must be a member of this group. Use the `usermod` command to add the user. For example, to add the user `stacker` to the `pvm_admin` group, execute:

```
sudo usermod -a -G pvm_admin stacker
```

The user must re-login for the change to take effect.

The NovaLink architecture is such that the compute driver runs directly on the PowerVM system. No external management element (e.g. Hardware Management Console or PowerVC) is needed. Management of the virtualization is driven through a thin virtual machine running on the PowerVM system.

Configuration of the PowerVM system and NovaLink is required ahead of time. If the operator is using volumes or Shared Storage Pools, they are required to be configured ahead of time.

Configuration File Options

The standard nova configuration options are supported. In particular, to use PowerVM SR-IOV vNIC for networking, the `pci_passthrough_whitelist` option must be set. See the networking-powervm usage devref for details.

Additionally, a `[powervm]` section is used to provide additional customization to the driver.

By default, no additional inputs are needed. The base configuration allows for a Nova driver to support ephemeral disks to a local volume group (only one can be on the system in the default config). Connecting Fibre Channel hosted disks via Cinder will use the Virtual SCSI connections through the Virtual I/O Servers.

Operators may change the disk driver (nova based disks - NOT Cinder) via the `disk_driver` property.

All of these values are under the `[powervm]` section. The tables are broken out into logical sections.

To generate a sample config file for `[powervm]` run:

```
oslo-config-generator --namespace nova_powervm > nova_powervm_sample.conf
```

The [powervm] section of the sample can then be edited and pasted into the full nova.conf file.

VM Processor Options

Configuration option = Default Value	Description
proc_units_factor = 0.1	(FloatOpt) Factor used to calculate the processor units per vcpu. Valid values are: 0 1.0
uncapped_proc_weight = 64	(IntOpt) The processor weight to assign to newly created VMs. Value should be be 1 and 255. Represents the relative share of the uncapped processor cycles the Virtu Machine will receive when unused processor cycles are available.

Disk Options

Configuration option = Default Value	Description
disk_driver = localdisk	(StrOpt) The disk driver to use for PowerVM disks. Valid options are: localdisk, ssp If localdisk is specified and only one non-rootvg Volume Group exists on one of the Virtual I/O Servers, then no further config is needed. If multiple volume groups exist, then further specification can be done via the volume_group_* options. Live migration is not supported with a localdisk config. If ssp is specified, then a Shared Storage Pool will be used. If only one SSP exists on the system, no further configuration is needed. If multiple SSPs exist, then the cluster_name property must be specified. Live migration can be done within a SSP cluster.
cluster_name = None	(StrOpt) Cluster hosting the Shared Storage Pool to use for storage operations. If none specified, the host is queried; if a single Cluster is found, it is used. Not used unless disk_driver option is set to ssp.
volume_group_name = None	(StrOpt) Volume Group to use for block device operations. Must not be rootvg. If disk_driver is localdisk, and more than one non-rootvg volume group exists across the Virtual I/O Servers, then this attribute must be specified.
volume_group_vios_name = None	(StrOpt) (Optional) The name of the Virtual I/O Server hosting the volume group. If this is not specified, the system will query through the Virtual I/O Servers looking for one that matches the volume_group_vios_name. This is only needed if the system has multiple Virtual I/O Servers with a non-rootvg volume group whose name is duplicated. Typically paired with the volume_group_name attribute.

Volume Options

Configuration option = Default Value	Description
fc_attach_strategy = vscsi	(StrOpt) The Fibre Channel Volume Strategy defines how FC Cinder volumes should be attached to the Virtual Machine. The options are: npiv or vscsi. It should be noted that if NPIV is chosen, the WWPNs will not be active on the backing fabric during the deploy. Some Cinder drivers will operate without issue. Others may query the fabric and thus will fail attachment. It is advised that if an issue occurs using NPIV, the operator fall back to vscsi based deploys.
vscsi_vios_connections_required = 1	(IntOpt) Indicates a minimum number of Virtual I/O Servers that are required to support a Cinder volume attach with the vSCSI volume connector.
ports_per_fabric = 1	(IntOpt) (NPIV only) The number of physical ports that should be connected directly to the Virtual Machine, per fabric. Example: 2 fabrics and ports_per_fabric set to 2 will result in 4 NPIV ports being created, two per fabric. If multiple Virtual I/O Servers are available, will attempt to span ports across I/O Servers.
fabrics = A	(StrOpt) (NPIV only) Unique identifier for each physical FC fabric that is available. This is a comma separated list. If there are two fabrics for multi-pathing, then this could be set to A,B. The fabric identifiers are used for the 'fabric_<identifier>_port_wwpns' key.
fabric_<name>_port_wwpns	(StrOpt) (NPIV only) A comma delimited list of all the physical FC port WWPNs that support the specified fabric. Is tied to the NPIV 'fabrics' key.

Config Drive Options

Configuration option = Default Value	Description
vopt_media_volume_group = root_vg	(StrOpt) The volume group on the system that should be used to store the config metadata that will be attached to the VMs.
vopt_media_rep_size = 1	(IntOpt) The size of the media repository (in GB) for the metadata for config driv Only used if the media repository needs to be created.
image_meta_local_path = /tmp/cfgdrv/	(StrOpt) The location where the config drive ISO files should be built.

Testing

Running Nova-PowerVM Tests

This page describes how to run the Nova-PowerVM tests. This page assumes you have already set up an working Python environment for Nova-PowerVM development.

With *tox*

Nova-PowerVM, like other OpenStack projects, uses tox for managing the virtual environments for running test cases. It uses Testr for managing the running of the test cases.

Tox handles the creation of a series of virtualenvs that target specific versions of Python.

Testr handles the parallel execution of series of test cases as well as the tracking of long-running tests and other things.

For more information on the standard tox-based test infrastructure used by OpenStack and how to do some common test/debugging procedures with Testr, see this wiki page:

https://wiki.openstack.org/wiki/Testr

PEP8 and Unit Tests Running pep8 and unit tests is as easy as executing this in the root directory of the Nova-PowerVM source code:

```
tox
```

To run only pep8:

```
tox -e pep8
```

To restrict the pylint check to only the files altered by the latest patch changes:

```
tox -e pep8 HEAD~1
```

To run only the unit tests:

```
tox -e py27,py34
```

Indices and tables

- *genindex*
- *modindex*
- *search*