Hewlett Packard
Enterprise

Helion OpenStack Carrier Grade 4.0

VNF INTEGRATION

16.10

Titanium Server

Cloud Administration, 16.10

Contents

1

Overview of VNF Integration

Overview of VNF Integration

This guide is provided to assist developers in deploying VNF applications in a Titanium Server environment.

The following VNF-related topics are included in this document:

- Supported Integration Points
  - Bootstrap Data
    - User Data
    - Config Drive
      - Injected Files
- Generating Ceilometer Statistics with cfn-init
- Guest VLANs
- Selected SDK Modules
  - VP KMOD
  - AVP PMD
  - Guest Heartbeat, Health Check, and Command Notification / Ack
  - Guest Resource Scaling
  - Guest Server Group Messaging

2

VNF Integration at Launch Time

You can use one or more mechanisms to pass information to a new VNF when it is originally launched. The VNF can use this information for system configuration purposes or to determine running options for the services it provides.

The mechanisms used to convey information from the Cloud OS to the VNF are the metadata server and config drive.

- user data is made available to the VNF through either the metadata service or the config drive
- cloud-init reads information from either the metadata server or the config drive

Detailed information about the OpenStack metadata server, config drive, and cloud-init is available online.

For more information, see http://docs.openstack.org.

Configuring Instances at Boot Time

Boot configuration user data can be passed to a virtual machine during startup.

For example, an Element Management System (EMS) may be used in cloud environments for VM configuration, but the VM may require some bootstrap information to successfully communicate with the EMS.

Titanium Server provides three mechanisms to accomplish this:

User Data

This is a mechanism for passing a local file to an instance when it is launched. This method is typically used to pass a shell script or configuration file.

To send user data when calling nova boot, use the **--user-data /path/to/filename** option, or use the Heat service and set the **user_data** property and **user_data_format** to **RAW**.

On initialization, the VM queries the metadata service through either the EC2 compatability API. For example:

```
$ curl http://169.254.169.254/openstack/2012-08-10/user_data
```

or the OpenStack metadata API. For example:

```
$ curl http://169.254.169.254/2009-04-04/user-data
```

In either case, text is returned to the VM and can be used for bootstrap configuration.

Access to the metadata server at 169.254.169.254 is provided by a virtual router attached to the tenant network on which the access request is made. Virtual routers are automatically configured as proxies to the metadata service.

The following requirements must be satisfied in order for a guest application to access the metadata service:

- There must be a route table entry to route traffic to the 169.254.169.254 address using a Neutron router, or using a suitable static route to the 169.254.169.254 address.
- The metadata server must know about the virtual machine instance associated with the MAC and IP addresses of the virtual Ethernet interface issuing the metadata service request. This is necessary for the metadata server to validate the request and identify the correct data to return to the VM.

On system managed subnets, the Neutron service has all address information associated with the virtual machines in its database.

On unmanaged subnets, you must tell the Neutron service the IP address of the network interface issuing the metadata service requests. This can only be done using the command line or API interfaces when instantiating the virtual machine. For example:

```
$ nova boot \
--nic net-id=net-uuid,vif-model=avp,v4-fixed-ip=172.18.0.1 \
my-new-vm
```

In this simple example, the option **v4-fixed-ip** tells Neutron the IP address for this interface. The interface's MAC address is automatically generated by Nova.

cloud-init

This is an open-source package available from https://cloudinit.readthedocs.org/en/latest/ that supports a variety of guests. It expects a particular file format for user data, retrieves the user data from the metadata server, and takes action based on the contents of the data.

Two commonly used input formats are:

shell scripts

You can configure an instance at boot time by passing a shell script as user data. The script file must begin with

```
#!
```

for **cloud-init** to recognize it as a shell script.

Cloud config files

A configuration format based on YAML that you can use to configure a large number of options on a system. For example, to set the hostname:

```
#cloud-config
hostname: mynode
fqdn: mynode.example.com
manage_etc_hosts: true
```

See https://cloudinit.readthedocs.org/en/latest for a complete list of capabilities.

The following requirements must be satisfied in order for a guest application to access the metadata service:

- There must be a route table entry to route traffic to the 169.254.169.254 address using a Neutron router, or using a suitable static route to the 169.254.169.254 address.
- The metadata server must know about the virtual machine instance associated with the MAC and IP addresses of the virtual Ethernet interface issuing the metadata service request. This is necessary for the metadata server to validate the request and identify the correct data to return to the VM.

Config drive

Titanium Server can be configured to use a special-purpose configuration drive (abbreviated *config drive*) to store metadata (including injected files). Metadata is written to the drive, which is attached to the instance when it boots. The instance can retrieve information normally available through the metadata service by reading from the mounted drive.

The config drive can be enabled by using the **--config-drive=true** option with nova boot.

The following example enables the config drive and passes user data, injecting two files and two key/value metadata pairs. These can be read from the config drive.

```
$ nova boot --config-drive=true --image my-image-name --key-name mykey \
--flavor 1 --user-data ./my-user-data.txt myinstance \
--file /etc/network/interfaces=/home/myuser/instance-interfaces --file known_hosts=/
home/myuser/.ssh/known_hosts \
--meta role=webservers --meta essential=false
```

From within the instance, the config drive volume is labeled **config-2**. You can mount the config drive as the **/dev/disk/by-label/config-2** device if your guest OS supports accessing disks by label. For example:

```
# mkdir -p /mnt/config
# mount /dev/disk/by-label/config-2 /mnt/config
```

The contents of the config drive depend on the options passed to nova boot. The contents of the config drive for the example above are:

```
ec2/2009-04-04/meta-data.json
ec2/2009-04-04/user-data
ec2/latest/meta-data.json
ec2/latest/user-data
openstack/2012-08-10/meta_data.json
openstack/2012-08-10/user_data
openstack/content
openstack/content/0000
openstack/content/0001
openstack/latest/meta_data.json
openstack/latest/user_data
```

For file format details and full details on config-drive see http://docs.openstack.org/user-guide/cli_config_drive.html.

⚠ **CAUTION:** If a VM uses config-drive for user data or file injection, VM evacuations due to a compute node failure and VM live migrations to another compute node will cause the config drive to be rebuilt on the new compute node and metadata to be populated, but user data and injected files are not populated in the evacuated or live-migrated config drive of the VM.

For a VM using **config-file** with file injection, it is recommended to copy the injected files to the root disk of the VM on initial boot, and to set a flag to prevent the use of injected files on subsequent boots.

Related Links

Customizing Images with User Data In a Heat Template

3

Accelerated Virtual Interfaces

VNFs can be configured to use accelerated virtual ports (AVP) that improve throughput when compared to emulated virtual drivers (example, e1000).

Titanium Server AVP virtual NIC is a shared memory based high performance networking device. Its potential maximum throughput is higher than other emulated virtual NIC devices (for example, e1000).

Titanium Server AVP Linux kernel device driver is delivered as source with the required makefiles in a compressed tarball, such that it can be compiled as an external kernel module.

Performance Enhanced Standard virtio - Support for vhost-net/vhost-user

Unmodified guests can use Linux networking and virtio drivers. This provides a mechanism to bring existing applications into the production environment immediately.

For virtio interfaces, Titanium Server supports vhost-user transparently by default. This allows QEMU and AVS to share virtio queues through shared memory, resulting in improved performance over standard virtio. The transparent implementation provides a simplified alternative to the open-source AVP kernel and AVP-PMD drivers included with Titanium Server. The availability of vhost-user also offers additional performance enhancements through optional multi-queue support for virtio interfaces.

For backwards compatibility, Guest OS can still be configured to leverage the open-sourceAccelerated Virtual Port (AVP-KMOD) drivers available at https://github.com/Wind-River/titanium-server.

wrs-avp-kmod—Accelerated Kernel Network Drivers

This component contains AVS-compatible kernel drivers for improved VM networking performance of kernel-based networking VNFs.

The Titanium Server AVP virtual network interface card (NIC) is a shared, memory-based, high-performance networking device. Its potential maximum throughput is higher than emulated virtual NIC devices (for example, e1000). This package provides the AVP Linux kernel device driver source. It can be compiled against most recent Linux kernel distributions.

The Titanium Server AVP Linux kernel device driver is delivered as source with the required makefiles in a compressed tarball so that it can be compiled for the applicable guest Linux distribution as an external kernel module.

wrs-avp-pmd—Accelerated DPDK Network Drivers

This component contains AVS-compatible DPDK drivers for high-performance DPDK-based networking VNFs.

The Titanium Server AVP virtual NIC is a high-performance networking device. It can provide line rate throughput (depending on the guest and AVS configuration). This package provides the Intel DPDK compatible Poll Mode Driver (PMD). It can be compiled as a component of an Intel DPDK distribution.

4

VNF Resource Scaling

You can autoscale VNF resources up/down by dynamically increasing or decreasing the number of virtual CPUs. This can be done to optimize the resource usage according to the application's demand for computational power.

Guest Server Scaling is a service to allow a guest to scale the capacity of a single guest server up and down on demand. Current supported scaling operation is CPU scaling.

The output of both the wrs-guest-scale SDK module and the wrs-server-group SDK module are required to be installed in a guest image for guest resource scaling.

wrs-guest-scale—VM Resource Scaling

VM Resource Scaling is a service to allow a guest to scale the capacity of a single guest server up and down on demand.

Currently, only scaling the number of online guest vCPUs is supported. The resources can be scaled up or down from the Nova CLI or the Titanium Server web administration interface. Scaling can also be set up using Heat to be automatically triggered based on Ceilometer statistics. This package contains an agent and APIs for integration with the Titanium Server Scale Up / Down service. These will handle the guest side of the coordinated efforts involved in scaling up and down guest resources.

Scaling Virtual Machine Resources

You can scale the resources of individual instances up or down.

Currently, the CPU resources for an instance are scalable.

For an instance to be scalable, the following requirements must be satisfied:

- The image used to launch the instance must support scaling.

  The example image provided with Titanium Server supports scaling. You can also build your own image, incorporating the required libraries and services. For more about building your own images, or about the technical requirements for scaling support, refer to the documentation included with the Titanium Server Software Development Kit.

- The flavor used to launch the instance must be configured with maximum and minimum scaling limits (the *scaling range*) for the resource.

When scaling a VM, use the credentials of the user that launched the VM. This ensures that quotas are correctly managed.

Depending on the resource being scaled, the scaling behavior applied by Titanium Server may be adjustable. For example, you can control which CPUs are released when the CPU resources for an instance are scaled down. To adjust scaling behavior, use the App Scale Helper script.

Normally, scaling is performed under the direction of Heat orchestration. For more about Heat autoscaling, see *Resource Scaling (Autoscaling)*. If required, you can scale resources manually from the command line using the **nova scale** command, with the following syntax:

```
~(keystone_admin)$ nova scale instance_id resource {up | down}
```

For example, to reduce the number of CPUs allotted to an instance, you can use this command:

```
~(keystone_admin)$ nova scale instance_id cpu down
```

NOTE: To scale up the resources for an instance, sufficient resources are required on the host. If all available resources are already allocated to other instances, you may need to free up resources manually.

The App Scale Helper Script

The App Scale Helper script supports resource scaling customizations.

This is an optional script running on the guest. If present, it can modify aspects of scaling behavior. For example, it can control which CPU is taken offline during a scale-down operation, overriding the default selection.

It can also call other scripts or programs. You can use this to coordinate or manage processing work for the vCPUs as you scale them.

For more about the App Scale Helper script, refer to the documentation for the *Titanium Server Software Development Kit*.

CPU Scaling

Titanium Server supports CPU up/down scaling for instances.

You can enable CPU scaling for an instance by setting a scaling range (see Setting the CPU Scaling Range on page 11). In addition, the **CPU Policy** must be set to **Dedicated**, and if the **CPU Thread Policy** is also present, it must be set to **Isolate** (see *Specifying Host CPU Threading Policies for a VM*).

When an instance is first started, all available vCPUs are brought online. If the instance is restarted, the number of online vCPUs is set to match the number when the instance was stopped. This is controlled by a helper script that automatically takes the required number of vCPUs offline.

When CPU resources are scaled up, a vCPU is brought online from the pool of available vCPUs for the instance, subject to the maximum allowed by the flavor. The lowest-numbered offline vCPU is selected.

When CPU resources are scaled down, a vCPU is taken offline, subject to the minimum allowed by the flavor. By default, the highest-numbered online vCPU is selected. This can be overridden using the App Scale Helper script.

For more about CPU scaling, refer to the documentation for the Titanium Server Software Development Kit.

NOTE: If there are insufficient resources to launch a scalable VM with the expected allotment when the VM is started or restarted, the launch fails. The VM is *not* automatically scaled down to compensate.

Setting the CPU Scaling Range

You can define the maximum and minimum CPU resources available to an instance by using a flavor.

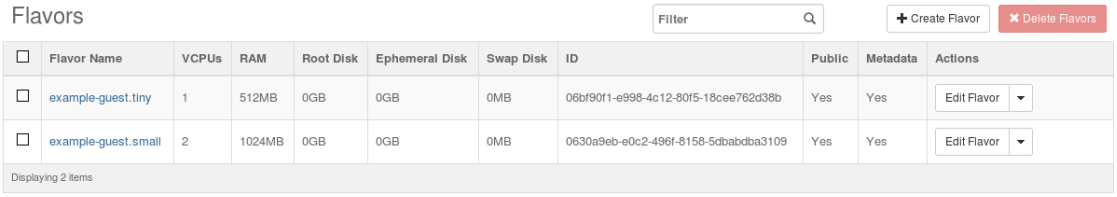
Prerequisites

For CPU scaling to take effect, the **CPU Policy** for the VM must be set to **Dedicated**. For more information, see *Specifying Host CPU Threading Policies for a VM*.

Procedure

1. Display the Flavors list.

   Select the **Admin** menu in the web administration interface, and then in the **System Panel** section of the menu, click **Flavors**.



2. Optional: Select a suitable flavor.

   You can edit it to specify the maximum and minimum vCPUs.

3. Specify the maximum number of vCPUs for the instance.

   In the Flavors list, locate the flavor, and then click **Edit** to open the Edit Flavor dialog box.

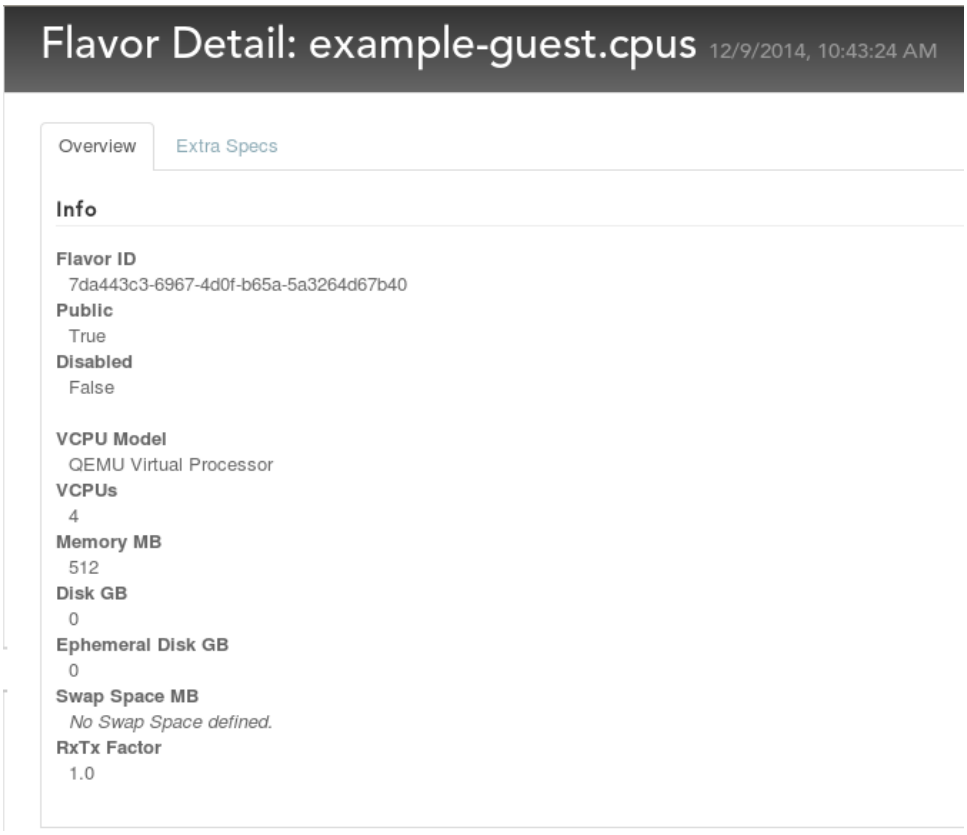
   In the **vCPU** field, enter the maximum number of vCPUs.

   When you are finished editing, click **Save** to return to the Flavors list.

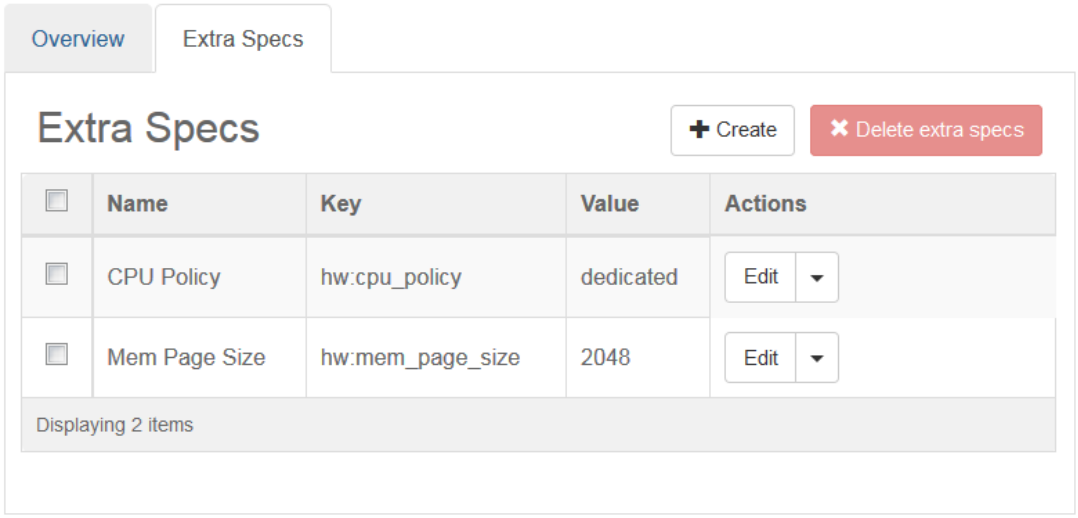
4. Specify the minimum number of vCPUs for the instance.

   You can specify the minimum number of vCPUs by adding an Extra Spec for the flavor.

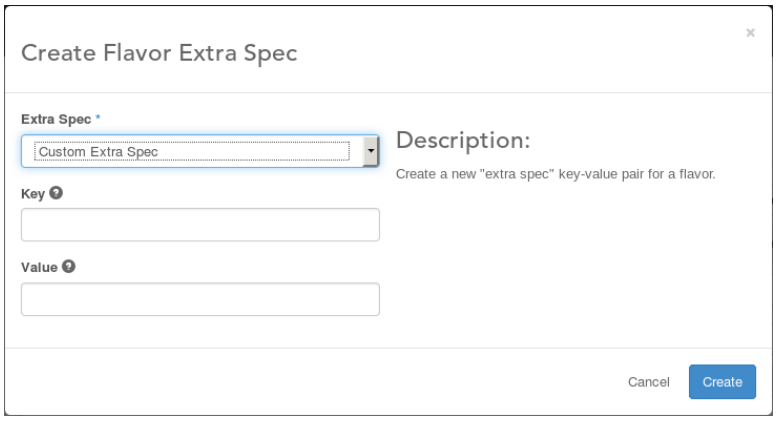
   a) In the Flavors list, click the **Flavor Name** for the flavor to open the Flavor Details dialog box.



   b) On the **Extra Specs** tab, click **Create**.



c) In the **Create Flavor Extra Spec** dialog, select **Minimum Number of CPUs** from the **Extra Spec** drop-down menu.



d) In the **Key** field, enter wrs:min_vcpus.

e) In the **Value** field, enter the minimum allowed number of vCPUs for the flavor.

f) Click **Create**.

5

High Availability Features

You can instrument Titanium Server-specific features in a VNF to further integrate it into the Titanium Server high-availability framework.

These features include:

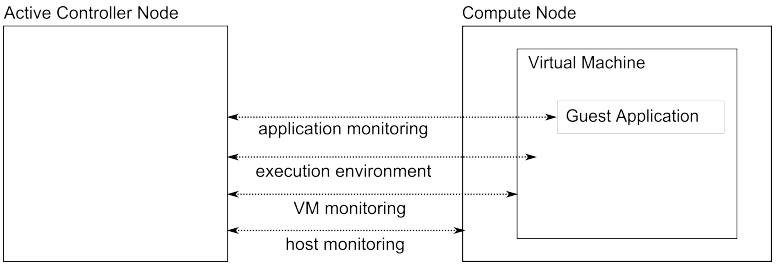
- Guest Heartbeat
- Server Group Messaging APIs

Virtual Machines and Carrier-Grade Availability

The Titanium Server virtualized environment provides a health monitoring mechanism that can be used to implement and support the deployment of guest applications in Carrier-Grade High Availability (HA) mode.

A simplified view of the health monitoring system is illustrated in the following figure:

Figure 1: Hardware and software health monitoring



Host monitoring

A host failure occurs when the compute node hardware that hosts the application fails, or when its kernel becomes unresponsive. Host failures are detected by the active controller node through an efficient and scalable monitoring process that runs continuously during normal operation of the Titanium Server cluster.

When such a failure is detected, Titanium Server automatically re-schedules all affected virtual machines for deployment on alternative compute nodes.

Host monitoring, and its recovery mechanisms, are always available for every deployed virtual machine in the Titanium Server Cluster.

Virtual machine monitoring

From the point of view of a guest application, a hardware failure occurs when the hosting virtual machine fails to execute. From the compute node, this means that the virtual machine process itself is experiencing execution problems, or that it is no longer running.

When such a failure is detected, Titanium Server automatically tries to restart the affected virtual machine on the same compute node. If the restart operation fails, the virtual machine is automatically scheduled to deploy on an alternative compute node.

Virtual machine monitoring, and its recovery mechanisms, are always available for every deployed virtual machine in the Titanium Server cluster.

Execution environment

The Titanium Server Guest-Client daemon, built into the guest image, verifies the execution environment. If the guest kernel becomes unresponsive, the lack of heartbeat messages from the daemon triggers an alarm on the active controller.

For all purposes, the virtual machine is considered then to be at fault, and is therefore re-scheduled for execution, first on the same compute node, and then on a different host if necessary.

Application monitoring

Software failures at the application level can be addressed within the context of the following failure scenarios:

Sudden death of the application process

The application process ends abruptly, likely because of a software bug in its code.

The application becomes unresponsive

Several conditions can lead the application process to stall, that is, to be unscheduled for additional work. This may happen because the application is waiting for some resource that is not available, or because of a bug in the application's logic.

The application declares itself to have failed

Logic built into the application determines conditions under which it must declare itself to be in an unrecoverable error state.

Handling of these cases is optional, and subject to the proper API integration into the guest application itself.

Application monitoring happens when the Titanium Server Guest-Client daemon in the virtual machine registers the application for monitoring. What happens after an application failure occurs is determined when the application is registered. By default, an application failure triggers a hard reboot of the virtual machine instance.

If the application makes use of the Titanium Server Guest Heartbeat API, it can instruct the active controller on how to proceed when the application declares itself in a state of error. This could include a restart of the virtual machine, which in the case of 1:1 HA application/VM pair, would trigger a VM switchover. The application can also use the *VM Peer Notification API* to receive notifications when virtual machines in the same server group go up or down.

Refer to the *Titanium Server SDK* for more information on how to configure the Guest-Client daemon and the use of the Titanium Server Guest Heartbeat API.

Data persistence is another aspect that impacts the high-availability of a running application. This is addressed by providing storage persistence across applications restarts for Cinder remote HA block storage using distributed storage backends, such as Ceph, for configuration databases.

By instrumenting the level of interaction between the guest application and the health-monitoring system, and by using the appropriate Cinder storage backends, guest applications can be deployed to run in different HA scenarios. Note that in all cases, monitoring of hardware and virtual machines, and their corresponding failure recovery mechanisms, are always active.

wrs-guest-heartbeat—Guest Heartbeat

This component contains APIs for integration with the Titanium Server Guest Heartbeat service.

Titanium Server Guest Heartbeat is a service to monitor the health of guest applications within a VM running under the Wind River Titanium Server. Loss of heartbeat will result in a specified corrective action (for example, rebooting the VM). Guest applications are given the opportunity to receive notification of, or even veto, actions affecting the VM. Guest applications can use this capability to cleanly shut down or even live migrate their service to a peer VM.

wrs-server-group—Server Group Messaging

This component contains APIs for the Titanium Server "server group" peer monitoring service.

Server Group Messaging is a service to provide simple, low-bandwidth datagram messaging and notifications for servers that are part of the same server group. This messaging channel is available regardless of whether IP networking is functional within the server, and it requires no knowledge within the server about the other members of the group. The service provides three types of messaging:

- Broadcast: enables a server to send a datagram (up to 3050 bytes) to all other servers within the server group
- Notification: provides servers with information about changes to the state of other servers within the server group
- Status: enables a server to query the current state of all servers within the server group (including itself)

This service is not intended for high-bandwidth or low-latency operations. It is best-effort and not guaranteed. For improved reliability, applications should do end-to-end ACKs and retries.

6

Data Reporting Using Ceilometer

You can instrument data-reporting features in your VNF to provide operational data to the Ceilometer service.

You can use Ceilometer and **cfn-init** to retrieve and parse data and **cfn-push-stats** to generate statistics that are application specific. These statistics can then optionally be used to generate Ceilometer Threshold Alarms and trigger Heat scaling operations.

7

Swift Object Storage Support

Systems with dedicated storage hosts can provide object storage using OpenStack Swift. VMs and system users can use this to store and exchange Ceph-backed files.

You can add Swift support from the command line at any time after installation. For more information, see *Titanium Server System Administration:Configuring Swift Object Storage*.

System administrators and tenant users can manage Swift containers and their contents from the CLI or the Web administration interface. VMs can use a Swift client or the Swift REST API to manage and access Swift containers and objects. For more information, consult the public OpenStack documentation.

VM Configuration for Access to Swift Object Storage

For access to Swift storage, you must attach the VM to an external network with a route to the controller OAM network. The VM uses a Swift client or Swift REST API to manage Swift containers and objects, sending commands to the Titanium Server Swift endpoint public URL. The Swift API service on the Titanium Server controller has access to the Swift containers and objects in the Ceph storage pool over the infrastructure network.

Sample Commands for Swift API Access from a VM

Some Swift API command examples using **curl** are provided for reference.

```
# show account stat (swift stat):
curl -i $publicURL -I -H "X-Auth-Token: $token"; echo

# show account details and list containers in JSON format (swift list):
```

```
curl -i $publicURL?format=json -X GET -H "X-Auth-Token: $token"; echo

# Create a container test1:
curl -i $publicURL/test1 -X PUT -H "Content-Length: 0" -H "X-Auth-Token: $token"; echo

# show containter
curl -i $publicURL/test1 -X GET -H "X-Auth-Token: $token"; echo

#upload a text file hello.txt which has length of 22 (output from ls -l) bytes.
curl -i $publicURL/test1/hello.txt -X PUT -H "Content-Length: 22" -H "Content-Type:
text/html; charset=UTF-8" -H "X-Auth-Token: $token" -T hello.txt

#upload a binary file CentOS-7-x86_64-GenericCloud.raw.tar.gz which has length of
358387943 (output from ls -l) bytes.
curl -i $publicURL/test1/CentOS-7-x86_64-GenericCloud.raw.tar.gz -X PUT -H "Content-
Length: 358387943" -H "X-Auth-Token: $token" -T CentOS-7-x86_64-
GenericCloud.raw.tar.gz;date

#download a text file and print out to console:
curl $publicURL/test1/hello.txt -X GET -H "X-Auth-Token: $token"

#download a binary file and save it to a local file with same name:
curl $publicURL/test1/CentOS-7-x86_64-GenericCloud.raw.tar.gz -X GET -H "X-Auth-Token:
$token" > CentOS-7-x86_64-GenericCloud.raw.tar.gz
```