



**Hewlett Packard  
Enterprise**

# Fundamentals of OpenStack® Technology

Student guide (2 of 2)

H6C68S E.01

Use of this material to deliver training without prior written permission from HPE is prohibited.

© Copyright 2017 Hewlett Packard Enterprise Development LP

The information contained herein is subject to change without notice. The only warranties for HPE products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

This is an HPE copyrighted work that may not be reproduced without the written permission of Hewlett Packard Enterprise. You may not use these materials to deliver training to any person outside of your organization without the written permission of HPE.

Microsoft®, Windows®, Windows NT® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

All other product names mentioned herein may be trademarks of their respective companies.

### **Export Compliance Agreement**

Export Requirements. You may not export or re-export products subject to this agreement in violation of any applicable laws or regulations.

Without limiting the generality of the foregoing, products subject to this agreement may not be exported, re-exported, otherwise transferred to or within (or to a national or resident of) countries under U.S. economic embargo and/or sanction including the following countries: Cuba, Iran, North Korea, Sudan and Syria. This list is subject to change.

In addition, products subject to this agreement may not be exported, re-exported, or otherwise transferred to persons or entities listed on the U.S. Department of Commerce Denied Persons List; U.S. Department of Commerce Entity List (15 CFR 744, Supplement 4); U.S. Treasury Department Designated/Blocked Nationals exclusion list; or U.S. State Department Debarred Parties List; or to parties directly or indirectly involved in the development or production of nuclear, chemical, or biological weapons, missiles, rocket systems, or unmanned air vehicles as specified in the U.S. Export Administration Regulations (15 CFR 744); or to parties directly or indirectly involved in the financing, commission or support of terrorist activities.

By accepting this agreement you confirm that you are not located in (or a national or resident of) any country under U.S. embargo or sanction; not identified on any U.S. Department of Commerce Denied Persons List, Entity List, US State Department Debarred Parties List or Treasury Department Designated Nationals exclusion list; not directly or indirectly involved in the development or production of nuclear, chemical, biological weapons, missiles, rocket systems, or unmanned air vehicles as specified in the U.S. Export Administration Regulations (15 CFR 744), and not directly or indirectly involved in the financing, commission or support of terrorist activities.

Printed in US

### **Fundamentals of OpenStack® Technology**

Student guide (1 of 2)

March 2017

---

# Contents

Module 9 – OpenStack® Object Storage (Swift).....	9-1
Learning objectives .....	9-2
Swift overview .....	9-3
Object Storage .....	9-4
Swift Functions & Features .....	9-5
Swift account, container, and object features.....	9-6
Object Storage (Swift) API version states as of Newton release .....	9-7
Swift architecture.....	9-8
Swift Ring – <i>Modified Consistent Hashing Ring</i> .....	9-10
Swift Ring Partition .....	9-11
Swift Ring lookup .....	9-12
Swift Object Addressing .....	9-13
Swift regions and zones .....	9-14
Swift Storage Policies.....	9-15
Swift Consistency Processes .....	9-16
Example Object PUT Request Flow.....	9-18
Example Object GET Request Flow.....	9-19
Adding storage in a ring .....	9-20
Common Swift Management tasks.....	9-21
Swift management operations.....	9-22
CRUD ( <i>Create, Read, Update, Delete</i> ) operations .....	9-23
Sample API PUT of an object.....	9-24
Swift from the Horizon UI .....	9-25
Troubleshooting Swift.....	9-26
Diagnosing Swift Issues .....	9-27
Troubleshooting with Swift log files .....	9-28
Helpful CLI commands .....	9-29
Module summary.....	9-30
Learning check .....	9-31
Module 10 – OpenStack® Orchestration Service (Heat) .....	10-1
Learning objectives .....	10-2
Heat overview.....	10-3
Heat Functionality.....	10-4
Heat terminology .....	10-5
Heat components .....	10-6
How a Heat template works .....	10-7
HOT ( <i>Heat Orchestration Template</i> ) .....	10-8
Heat template components .....	10-9
Sample Heat template format.....	10-10

Heat stack .....	10-13
Orchestration (Heat) API version states as of Newton release .....	10-14
Common Heat management tasks.....	10-15
Obtaining values required for the Heat template.....	10-16
Launching a stack from the command line.....	10-17
Viewing Stack Details from the CLI .....	10-18
Launching a stack from the Horizon UI .....	10-19
Viewing Stack Details from the Horizon UI.....	10-22
Troubleshooting Heat.....	10-23
Troubleshooting .....	10-24
Module summary.....	10-25
Learning check.....	10-26
 Module 11—OpenStack® Telemetry Service (Ceilometer) .....	11-1
Learning objectives .....	11-2
Ceilometer overview.....	11-3
Ceilometer Functionality.....	11-4
Common Ceilometer Use Cases.....	11-5
Ceilometer metering .....	11-6
Ceilometer Terminology .....	11-7
Telemetry (Ceilometer) API version states as of Newton release.....	11-9
Ceilometer architecture .....	11-10
Ceilometer Architectural Components.....	11-11
Processing the Ceilometer data .....	11-13
Ceilometer Data Collection.....	11-14
Transforming the Ceilometer Data ( <i>Optional</i> ) .....	11-15
Publishing the Ceilometer Data.....	11-16
Storing the Ceilometer Data .....	11-17
Ceilometer alarms (Aodh) .....	11-18
Common Ceilometer management tasks .....	11-19
Common CLI tasks (1 of 2) .....	11-20
Common tasks (2 of 2).....	11-21
Ceilometer Info in the Horizon UI .....	11-22
Ceilometer troubleshooting .....	11-23
Ceilometer logging .....	11-24
Module summary.....	11-25
Learning check .....	11-26
 Module 12 – Other OpenStack Projects .....	12-1
Learning objectives .....	12-2
OpenStack Service Course Coverage .....	12-3
CloudKitty (Chargeback and Billing).....	12-4
The state of chargeback and billing in OpenStack® .....	12-5
Barbican (Key management).....	12-6
What is Barbican? .....	12-7
Designate (DNS as a Service) .....	12-8
What is Designate? .....	12-9
Freezer (Backup as a Service).....	12-10
Backup as a Service architecture.....	12-11
Backup service features .....	12-12
Example usage .....	12-13

Fuel (OpenStack® Environment Provisioning).....	12-14
What is the OpenStack® Fuel project? .....	12-15
Fuel architecture .....	12-16
Ironic (OpenStack® Bare Metal Provisioning).....	12-17
Ironic overview .....	12-18
Use cases for OpenStack® Bare Metal Provisioning .....	12-19
The OpenStack® Ironic logical architecture .....	12-20
Key technologies for baremetal hosting .....	12-21
Ironic deployment architecture .....	12-22
Prerequisites for baremetal deployment.....	12-23
Baremetal deployment steps.....	12-24
Magnum (Container Virtualization).....	12-25
What is Magnum? .....	12-26
Magnum architecture.....	12-27
Manila (Shared File System).....	12-28
What is Manila? .....	12-29
Sahara (Data Processing) .....	12-30
Sahara Functionality.....	12-31
Sahara interactions with other OpenStack Projects .....	12-32
Trove (Database as a Service).....	12-33
Trove Functionality .....	12-34
Trove use cases .....	12-35
Trove terminology.....	12-36
Trove architecture overview .....	12-37
Trove components.....	12-38
Trove supported Databases .....	12-39
Zaqar (Multi-tenant cloud messaging).....	12-40
What is Zaqar? .....	12-41
Zaqar design principles .....	12-42
 Module 13 – OpenStack Deployment Planning .....	13-1
Learning objectives .....	13-2
OpenStack deployment considerations .....	13-3
Cloud controller .....	13-4
Cloud controller hardware considerations .....	13-5
Separating services .....	13-6
HPE cloud offerings .....	13-7
The HPE POV: Hybrid delivery with an HPE Cloud .....	13-8
HPE Converged Cloud Professional Services .....	13-9
HPE Converged Cloud management offerings .....	13-10
HPE Converged Cloud Professional Services portfolio.....	13-11
Module summary .....	13-12
Learning check .....	13-13
Take control of your future with training from HPE Education Services.....	13-18



Hewlett Packard  
Enterprise

# Fundamentals of OpenStack® Technology

Module 9—OpenStack® Object Storage (Swift)

H6C68S E.01

© Copyright 2017 Hewlett-Packard Enterprise Development LP

## Learning objectives

After completing this module, you should be able to:

- Define object storage
- Discuss the primary function and features of OpenStack® Swift
- Explain the features of Swift accounts, containers, and objects
- Describe the Swift architecture
- Explain the operation of the Swift ring
- Describe the purpose of Swift zones and regions
- Discuss the data consistency features of Swift
- Complete some common Swift management tasks and troubleshoot common Swift issues



© Copyright 2017 Hewlett Packard Enterprise Development LP

2

## Learning objectives

After completing this module, you should be able to:

- Define object storage
- Discuss the primary function and features of OpenStack® Swift
- Explain the features of Swift accounts, containers, and objects
- Describe the Swift architecture
- Explain the operation of the Swift ring
- Describe the purpose of Swift zones and regions
- Discuss the data consistency features of Swift
- Complete the more common Swift management tasks and troubleshoot common Swift issues



# Swift overview

## Object storage

- A storage architecture that manages massive amounts of data
- Data is stored in the form of objects instead of the conventional file or block storage architectures
- Objects includes a variable amount of metadata, data, and a globally unique identifier
- Uses a namespace that can span multiple instances of physical hardware to achieve replication
- Uses ordinary hardware and hard drives
- Best storage for data that does not change frequently:
  - Backups
  - Archives
  - Video and audio files
  - Virtual machine images



© Copyright 2017 Hewlett Packard Enterprise Development LP

4

### Object storage

Object storage is a storage architecture that manages massive amounts of data as objects instead of the conventional file or block storage architectures. Objects includes a variable amount of metadata, data, and a globally unique identifier.

Object storage uses a namespace that can span multiple instances of physical hardware, and data management functions like data replication and data distribution at object-level granularity. It provides relatively inexpensive, scalable, and self-healing retention of massive amounts of unstructured data.

Reliability is achieved on ordinary hardware and disk drives by replicating objects across multiple servers and locations. Object storage is unsuitable for data that changes frequently because data consistency is achieved only eventually, there may be a delay before a request returns the latest version of the data.

Object storage is the best storage for data that does not change much, like backups, archives, video and audio files, and virtual machine images.

Retrieving data from object storage takes more time than accessing a file from NAS. In addition to having significantly slower throughput than a traditional file system, the other big drawback of object storage is that data consistency is achieved only eventually. Whenever you update a file, you might have to wait until the change is propagated to all of the replicas before requests return the latest version. This makes object storage unsuitable for data that changes frequently.

“Unstructured data store” refers to a storage that stores bits. Object storage stores BLOBs (Binary Large Object) of data.

## Swift functions and features

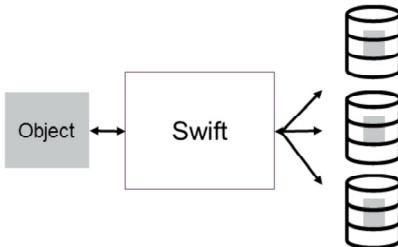
Reliably store billions of objects distributed across standard hardware

### General features

- Allows you to store and retrieve data objects by URL over any IP network
- Does not use a conventional file system
- Data is distributed evenly throughout the system
- Account, container, and object structure
- No central database
- Accessed with a RESTful API
- Provides data redundancy (at least 3 replicas)
- Auditors to check staleness of data
- Eventual consistency

### Hardware features

- Runs on commodity hardware
- Hardware-agnostic (no need for RAID)
- Supports multi-tenancy
- Augments SAN/NAS/DAS
- Optimized for scaling to zettabytes



### Swift functions and features—Reliably store billions of objects distributed across standard hardware

Swift is open source software used for managing scalable data storage using clusters of commodity hardware to store large amounts of data. Swift uses a distributed architecture to provide scalability and redundancy. Internally, Swift writes objects to multiple hardware devices (called “nodes”), and Swift ensures data replication and integrity across the cluster. Since Swift manages the replication and distribution of objects across different devices, inexpensive commodity hard drives and servers can be used.

Swift achieves high scalability by relaxing constraints on consistency. Listings and aggregate metadata (like usage information) might not be immediately accurate. Similarly, reading an object that has been overwritten with new data may return an older version of the object data. Swift provides the ability for the client to request the most up-to-date version at the cost of request latency.

Upgrades and cluster resizes can be easily performed on a production cluster with zero end-user downtime. Swift provides both forward and backwards compatibility of its API, so a Swift cluster can be running multiple versions of the Swift software at the same time, as is common while the software is being upgraded. During the process of resizing, the incongruent data about where data lives is simply seen as a failure and the replication process ensures that the data will be moved to its correct location.

## Swift account, container, and object features

### Account

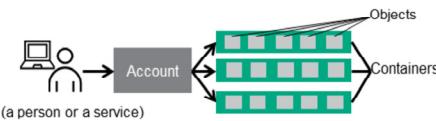
- Commonly compared to a filesystem volume
- Has a database that stores both the account's metadata and a listing of all its containers
- Can be designated for single-user, multi-user, or service (e.g. backup) access
- Does not contain objects or object metadata
- A storage account not a user identity

### Container

- User-defined storage compartments for objects
- Similar to an OS folder but cannot be nested
- Unlimited number of containers per account
- All objects are stored in precisely one container
- Has a database that stores both the container's metadata and a record for each object it contains
- Does not contain actual objects or their metadata

### Object

- Actual data stored in Swift (e.g. photos, docs, DB, backups, filesystem snapshots)
- Data is stored as is (no compression or encryption)
- Uploaded data consists of:
  - A location (container)
  - Object name
  - Data
  - Optional, user-specified metadata consisting of key/value pairs



 Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

6

## Swift account, container, and object features

Swift manages mapping between the names of entities stored on disk and their physical location using entity called "the ring." There are separate rings for accounts, containers, and one object ring per storage policy. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine its location in the cluster.

An **account** organizes and stores objects in object storage. It is similar to the concept of a Linux directory, but it cannot be nested.

A **container** is a storage compartment for your objects and provides a way for you to organize your objects. Containers cannot be nested. You can, however, create an unlimited number of containers within your account. Data must be stored in a container, so you must have at least one container defined in your account prior to uploading data. When it comes to container names, the only restriction is that they cannot contain a forward slash (/) and must be less than 256 bytes in length. The length restriction applies to the name after it has been URL-encoded. For example, a container name of "Course Docs" would be URL-encoded as Course%20Docs. Therefore, it would be 13 bytes in length rather than the expected 11.

An **object** is the basic storage entity and any optional metadata that represents a file you store in Swift. When you upload data to OpenStack® Object Storage, that data is stored as is (no compression or encryption) and consists of a location (container), the object's name, and any metadata consisting of key/value pairs. For example, you can chose to store a backup of your favorite OpenStack® training course and organize it into folders. In this case, each object could be tagged with metadata such as ILT: OpenStack® Fundamentals, ILT: OpenStack® Networking.

The maximum allowable size for a storage object upon upload is 5 gigabytes (GB) and the minimum is zero bytes. You can use the built-in large object support and the Swift utility to retrieve objects larger than 5GB.

In summary:

- Each object is stored in a container.
- One or more containers can be owned by an account.
- Each object is identified by its account, container, and object ID.

## Object Storage (Swift) API version states as of Newton release

- Compute API v1 (**current**)



© Copyright 2017 Hewlett Packard Enterprise Development LP

7

## Storage (Swift) API version states as of Newton release

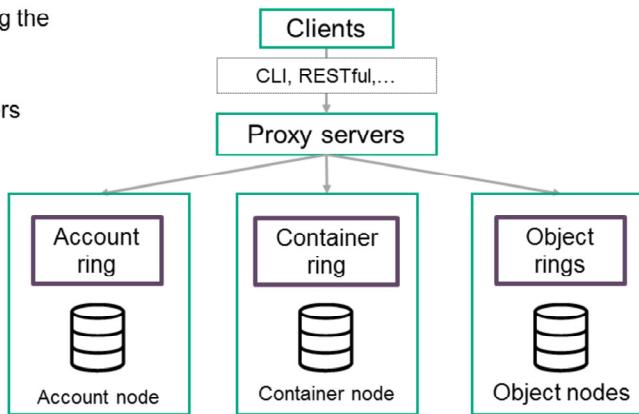
OpenStack® Object Storage API v1 is in current state for the Newton release. There are no release notes available for object storage in Newton release.



## Swift architecture

## Swift architecture

- A **Swift cluster** is a group of nodes (systems) running the full set of processes and services need to act as a distributed storage system
- An **account ring** is responsible for listing of containers
- A **container ring**:
  - Is a directory of the containers the objects belong to
  - Tracks statistics on objects
- **Proxy servers**:
  - Coordinate the read-write requests from clients
  - Implement the read-write guarantees of the systems
  - Do not cache objects (proxy them directly)
  - Are CPU- and network I/O-intensive
- An **object ring** is a directory containing the list of storage drives that are used to store specific objects and their associated metadata



 Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

9

## Swift architecture

The Swift architecture is well-distributed to prevent any single point of failure as well as to scale horizontally. It includes the following components:

- **Proxy server** (`swift-proxy-server`) accepts incoming requests through the OpenStack® Object API or just raw HTTP. It accepts files to upload, modifications to metadata, or container creation. In addition, it serves files or container listing to web browsers. The proxy server can use an optional cache (usually deployed with memcache) to improve performance.
- **Account servers** manage accounts defined with the Object Storage service.
- **Container servers** manage a mapping of containers (folders) within the Object Storage service.
- **Object servers** manage actual objects (files) on the storage nodes.

Several periodic processes perform housekeeping tasks on the large data store. The most important of these are the replication services, which ensure consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.

Authentication is handled through configurable WSGI middleware (which is usually Keystone). The proxy server is responsible for tying together the rest of the OpenStack® Object Storage architecture. For each request, it looks up the location of the account, container, or object in the hash ring (see below) and routes the request accordingly. The public API is also exposed through the proxy server.

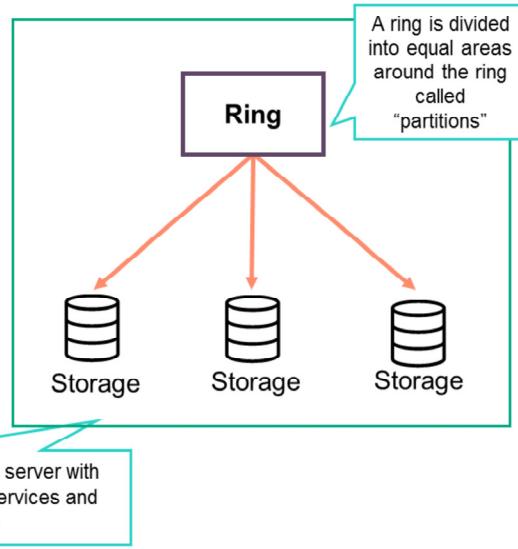
The object server is a very simple BLOB storage server that can store, retrieve, and delete objects stored on local devices. Objects are stored as binary files on the file system with metadata stored in the file's extended attributes (xattrs). This requires that the underlying file system choice for object servers support xattrs on files. Some file systems, like ext3, have xattrs turned off by default.

Each object is stored using a path derived from the object name's hash and the operation's timestamp. Last write always wins and ensures that the latest object version will be served. A deletion is also treated as a version of the file (a 0 byte file ending with ".ts," which stands for "tombstone"). This ensures that deleted files are replicated correctly and older versions do not reappear in failure scenarios.

## Swift ring

### Modified consistent hashing ring

- A directory structure used by the Swift proxy server to determine the location of accounts, containers, and objects on the various storage disks of a Swift cluster
- There are separate rings for accounts, containers, and one object ring per storage policy
- Divided into partitions, each addressable by the hashed value of the storage location
- Data replicated across nodes—Avoids SPOF
- Created and updated with the `swift-ring-builder` utility



Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

10

### Swift ring—Modified consistent hashing ring

A **ring** represents a mapping between the names of entities stored on disk and their physical location. A ring maintains this mapping using zones, devices, partitions, and replicas. Each partition in the ring is replicated, by default, 3 times across the cluster, and the locations for a partition are stored in the mapping maintained by the ring. The partitions of the ring are equally divided among all the devices in the Swift installation.

Swift ring determines where data should reside in the cluster. It maintains this mapping using zones, devices, partitions, and replicas. A copy of the ring should reside on every node of the cluster.

Rings are statically built and assigned as part of post installation tasks. They use a gzipped data structure. There are separate rings for account databases, container databases, and objects.

When it comes to ring management, remember that servers do not modify the rings—they are externally managed. Also, they are built and managed by the `ring-builder` utility.

## Swift ring partition

- A Swift ring partition:
  - Is a segment of the Swift ring that indicates the storage systems storing the account, container, or object data
  - Maps to multiple storage drives, each of which are used to store the account, container, or object data written to it
  - Is a section of address space, not disk space
  - Is not related to the partitioning of a hard drive
- A ring is evenly divided into partitions during the ring creation process
- Number of partitions is determined by  $2^n$ , where  $n =$  partition power
  - For example, if  $n = 10$ , there would be 1024 partitions created on the ring
- Each partition has the following info for each replica's storage device:
  - Zone
  - Weight
  - IP address of the server containing the device
  - The TCP port the server listens on for device requests
  - Device ID
  - Metadata info



© Copyright 2017 Hewlett Packard Enterprise Development LP

11

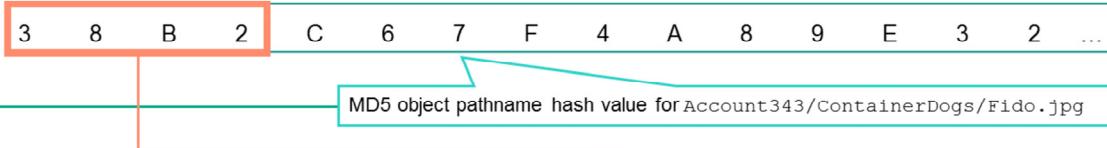
## Swift ring partition

The partitions of the ring are equally divided among all the devices in the Swift installation. When you need to move partitions around (for example, if a device is added to the cluster), the ring ensures that a minimum number of partitions are moved at a time, and only one replica of a partition is moved at a time.

Weights can be used to balance the distribution of partitions on drives across the cluster. This can be useful, for example, when different-sized drives are used in a cluster.

## Swift ring lookup

- 1. Partition key size :** Length of MD5 pathname hash to use as the partition key, which is determined when the ring is created using  $2^n$ ; here n is the hash size (e.g. a hash value of 4 defines a key size of 16 bits)



- 2. Partition/replica to device table:** A two dimensional table that maps each partition-key (x axis)/replica-number (y axis) pair to a storage device ID

	38B0	38B1	38B2	...
0	8	2	1	...
1	2	1	8	...
2	1	8	2	...

----->

- 3. Device table:** Maps each storage device ID to a location (e.g. IP address, TCP port, device file, etc.) partition key

ID1	15.168.97.21:6003/dev/sda10
ID8	15.168.22.4:6003/dev/sda7
ID2	15.168.13.1:6003/dev/sda5

Device table for partition 38B2 lists 3 replica storage devices and the information necessary to send or retrieve data from them (e.g. Replica #1 of Partition 38B2 is located on Device 8)

## Swift ring lookup

The ring uses a configurable number of bits from a path's MD5 hash as a partition key that designates a set of devices (based on the replication factor). The number of bits kept from the hash is the partition key length.

## Swift object addressing

### Examples

You access an object by its storage location and do not need to know where in the Swift cluster the object is physically located

#### – Object address:

sdb1/objects/343/ded/abfa65c7ed96b09251b0600c39eccded/1378769848.49243.data

#### – Account address:

sdb1/accounts/225/213/70d72c1ef3561f867d35913daeb45213/70d72c1ef3561f867d35913daeb45213.db

#### – Container address:

sdb1/containers/253/a8d/7e84c5bddfc36410f579aebd4378a8d/7e84c5bddfc36410f579aebd4378a8d.db

## Swift object addressing

Swift uses the underlying file system to store data on disk. An administrator can use usual file system tools to find and inspect this data. Swift uses the following convention for storing objects:

/path\_to\_mount\_points/device/objects/partition/hash\_suffix/hash/

Accounts and containers are stored similarly, with the `objects` part of the path replaced with `accounts` or `containers`.

The directory is where all data for the object is stored. Inside the directory, there is usually just one file (named `<timestamp>.data`). The object data is stored in the file, and the object metadata is stored in the extended attributes (`xattrs`) of the file.

If you delete the object, the `.data` file is deleted and a `<timestamp>.ts` ("ts" stands for "tombstone") file is created as a zero-byte file. This is a delete marker that will be eventually reaped, but it exists to ensure that the delete properly propagates to all replicas in the cluster.

Swift uses this scheme of timestamps in the file name to implement conflict resolution. Each piece of data is assigned a timestamp by the proxy server when the request comes to the cluster, and that timestamp is used to name the file on disk on the object (or account or container) server. Usually, there will be a single file in the directory. If Swift receives multiple, concurrent requests to write to the same object, multiple `.data` files can be written to this directory. Swift uses a "last-write-wins" policy to resolve such conflicts, choosing the most recent file by timestamp.

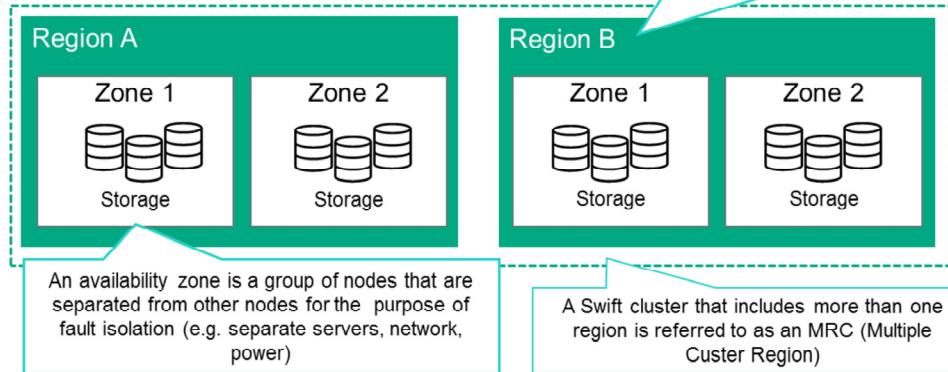
Here is a full DevStack path example:

`/opt/stack/data/swift/1/sdb1/objects/343/ded/abfa65c7ed96b09251b0600c39eccded/1378769848.49243.data`

## Swift regions and zones

- Partitions can be replicated across zones and regions
- Lowest latency placement algorithm is used

A region is generally used to geographically group portions of a Swift cluster for the purpose of management granularity or set storage policies by latency to a region



### Swift regions and zones

**A zone** is a set of distinct storage hardware isolated from other zones. This provides fault isolation. A zone is a group of nodes that is as isolated as possible from other nodes (separate servers, network, power, and even geography). The ring guarantees that every replica is stored in a separate zone.

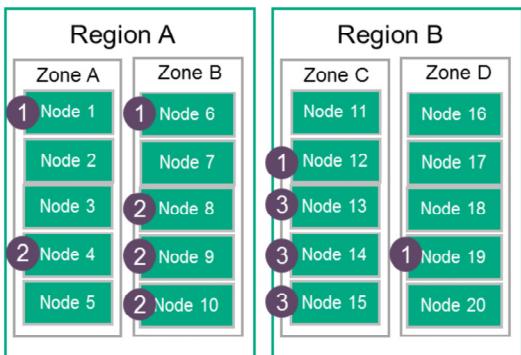
**A region** is a portion of the cluster that is typically located remotely from the data center. This allows for management of higher latency copies of partitions.

The combination of Swift pieces allows a Swift cluster to be highly fault-tolerant.

Availability zones within a single geographic region enable data to be written to hand-off nodes, if primary nodes in that zone are not available.

Swift is able to successfully survive hardware failures, including the loss of an entire availability zone with no impact to the end user.

## Swift storage policies



- 1 Storage policy 1 (distributed across multiple regions)
- 2 Storage policy 2 (distributed across a single region)
- 3 Storage policy 3 (distributed across fast media (e.g. SSDs))

- Enable Swift operators to define storage space with a cluster-based, specific data storage requirements, such as:
  - Location
  - Replication
  - Hardware
- Set at the object level and implemented at the container level
- Use cases:
  - Distribute data across multiple regions
  - Distribute replicas across a lower latency region
  - Ensure that faster disk technology drive (e.g. SSD) is used for specific type of data

## Swift storage policies

Storage policies let Swift operators define space within a cluster that can be customized in numerous ways, that is, by location, replication, hardware, and partitions, to meet specific data storage needs.

To implement storage policies, you must create them in the cluster and then apply them to containers. Creating user-defined storage policies is a two-step process:

1. The policies are declared in the `swift.conf` file with a name and an index number.
2. The policies' index number is used to create the corresponding builder files and rings.

Swift defaults to an internally referenced storage policy 0, which is applied to all containers. There is always a minimum of one storage policy per cluster as a result. In addition to being the system default, this storage policy 0, named "Policy-0" allows for backward compatibility with clusters created before the storage policies feature was added to Swift.

## Swift consistency processes (1 of 2)

- Responsible for managing the consistency of data caused by data corruption, hardware failures, and data deletion
- It consists of the following components: auditors, updaters, a replicator, an account reaper
- **Auditors:**
  - Continually scan disks to ensure that the data stored on a disk has not suffered any bit-rot or file system corruption
  - If an error is found, the corrupted object is moved to a quarantine area and data is replaced with a known good copy through replication
- **Updaters:**
  - Are responsible for keeping the object listings in the containers correct and keeping the account listings up-to-date
  - Update the object count and bytes used in the container metadata, as well as the object count, the container count, and bytes used in the account metadata



© Copyright 2017 Hewlett Packard Enterprise Development LP

16

## Swift consistency processes (1 of 2)

Several periodic processes perform housekeeping tasks on a large data store:

- **Auditors** crawl the local server checking the integrity of the objects, containers, and accounts. If corruption is found, the file is quarantined, and replication replaces the bad file from another replica. If other errors are found, they are logged (for example, when an object's listing cannot be found on any container server).
- **Updaters** update container and account data. The storing of data and the creation of an object listing are two different things. There are times when such data cannot be immediately updated, such as during a failure or a period of high load (for example, if a container server is under load and a new object is put into the system). The object is immediately available for reads as soon as the proxy server responds to the client with success. However, the container server does not update the object listing, and so the update is queued for a later update. Container listings, therefore, might not immediately contain the object.

## Swift consistency processes (2 of 2)

**– A replicator:**

- Is responsible for maintaining replicated copies of partitions
- Continuously compares each partition with its copies
- Ensures that the data stored in the cluster is where it should be
- Ensures that enough copies of data exist in the system
- Is responsible for repairing any corruption or degraded durability in the cluster

**– An account reaper:**

- Scans the account server occasionally for account databases marked for deletion
- Removes data from deleted accounts in the background
- Is only triggered on accounts where that server is the primary node, so that multiple account servers are not all trying to do the same work at the same time



© Copyright 2017 Hewlett Packard Enterprise Development LP

17

## Swift consistency processes (2 of 2)

**A replicator** is designed to keep the system in a consistent state in the face of temporary error conditions like network outages or drive failures. The replication processes compare local data with each remote copy to ensure that they all contain the latest version. Object replication uses a hash list to quickly compare subsections of each partition, and container and account replication uses a combination of hashes and shared high water marks.

A replicator also ensures that data is removed from the system. When an item (an object, a container, or an account) is deleted, a tombstone is set as the latest version of the item. The replicator sees the tombstone and ensures that the item is removed from the entire system.

The **account reaper** removes data from deleted accounts in the background. The account reaper runs on each account server and occasionally scans the server for account databases marked for deletion. It only triggers on accounts that the server is the primary node for.

An account is marked for deletion by entering a `DELETE` request on the account's storage URL. This puts the value `DELETED` into the `status` column of the `account_stat` table in the account database (and replicas), indicating the data for the account should be deleted.

The deletion process for an account itself is straightforward. For each container in the account, first each object is deleted and then the container is deleted. Any deletion request that fails does not stop the overall process, but it causes the overall process to fail eventually (for example, if an object delete times out, the container is not able to be deleted later, and therefore, the account is not deleted either). The overall process continues even on a failure so that it doesn't get hung up reclaiming cluster space because of one troublesome spot. The account reaper repeatedly tries to delete an account, until it eventually becomes empty, at which point the database reclaim process within the `db_replicator` eventually removes the database files.

## Example object PUT request flow

1. The client uses the Swift API to make an HTTP request to put an object into an existing container
2. The proxy server process determines which data stores the object will be stored to and the object ring to be used
3. The account, container, and object name value are hashed to determine the ring partition where the directory data for the object is stored
4. The drive IDs, their location (e.g. IP address), and similar, are added to the partition tables
5. The object's data and any associated metadata are copied concurrently to object storage drives (if one of the primary storage nodes is unavailable, the proxy chooses an appropriate hand-off node to write data to)
6. When a majority of the object servers respond successfully, the proxy returns a success message to the client

## Example object PUT request flow

Six steps on the slide above describe the example object PUT request flow. PUT requests are used to store data in the Swift cluster.

## Example object GET request flow

1. The client uses the Swift API to make an HTTP request to get the /account/container/object
2. The proxy server hashes the /account/container/object URL and uses the value to determine where the object is stored from the appropriate rings
3. Proxy server sends a request to the storage devices to fetch the requested object
4. The proxy server receives the object from the first storage device to respond successfully and provides it to the client



© Copyright 2017 Hewlett Packard Enterprise Development LP

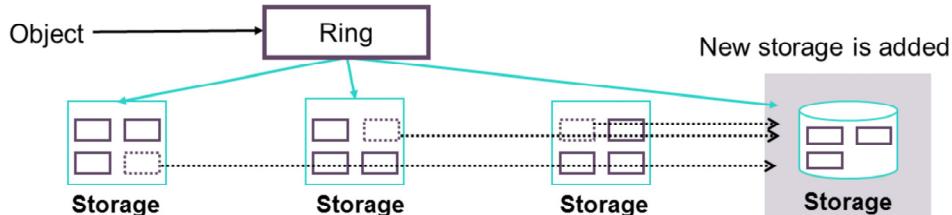
19

## Example object GET request flow

Four steps on the slide above describe the example object GET request flow. GET requests are used to retrieve data from the Swift cluster.

## Adding storage in a ring

- Swift Rings are manually constructed by administrators based on the Swift configuration desired
- When new storage is added, the ring tables must be updated and partitions must be redistributed across all available storage
  - `$ swift-ring-builder <builder_file> add z<zone>-<ip>:<port>/<device_name>_<meta> <weight>`
  - `$ swift-ring-builder <builder_file> rebalance`
- Once the administrator has installed new ring files, Swift carries out the partition redistribution



 Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

20

## Adding storage in a ring

Whenever you add additional hardware to Swift, it automatically redistributes partitions, so that the load is balanced across all arrays.

The OpenStack® administration documents recommend that there be a minimum of 100 partitions per drive to ensure even distribution across the drives. A good starting point might be to figure out the maximum number of drives the cluster will contain, multiply by 100, and then round up to the nearest power of two (partition key sizes are always a power of 2).

It is also recommended to keep the number of partitions small (relatively). The more partitions there are, the more work has to be done by the replicators and other backend jobs and the more memory the rings consume.

Configuring five zones and a replication factor of three are also well-tested recommendations.

In the example on the slide above, the `<builder_file>` is the name of the original ring builder file created, `<zone>` is the number of the zone this device is in, `<ip>` is the IP address of the server the device is in, `<port>` is the port number that the server is running on, `<device_name>` is the name of the device on the server (for example: `sdb1`), `<meta>` is a string of metadata for the device (optional), and `<weight>` is a float weight that determines how many partitions are put on the device relative to the rest of the devices in the cluster (for example, 100 times the number of TB on the drive).



## Common Swift management tasks

## Swift management operations

- Common administrative operations include:
  - Managing the rings
    - Add and remove storage devices
    - Add and remove servers
    - Search rings for devices
    - Rebalance rings
  - System updates
    - Recommendation: Perform updates on one zone at a time
    - Let the updated zone run for a day before updating another zone
  - Drive failures
    - Unmount the failed drive and leave it unmounted until a replacement drive arrives



© Copyright 2017 Hewlett Packard Enterprise Development LP

22

### Swift management operations

You can build the storage rings on any server with the appropriate version of Swift installed. When storage rings are built or changed (rebalanced), you must distribute the rings to all the servers in the cluster. Storage rings contain information about all the Swift storage partitions and how they are distributed between the different nodes and disks.

#### System updates

It is recommended that system updates and reboots be performed on one zone at a time. This enables the Swift cluster to stay available and responsive to requests. After updating a zone, let it run for a while before updating another zone to ensure that it is operating properly.

#### Drive failures

In the event that a drive has failed, the first step is to make sure the drive is unmounted. This makes it easier for Swift to work around the failure until it has been resolved.

If the drive will be replaced immediately, then it is best to replace the drive, format it, remount it, and let replication fill it up.

If the drive cannot be replaced immediately, then it is best to leave it unmounted and remove the drive from the ring. This allows all the replicas that were on that drive to be replicated elsewhere until the drive is replaced. After the drive is replaced, it can be re-added to the ring.

## CRUD (create, read, update, delete) operations

Four basic functions of persistent storage

Operation	Command
Post (create or update)	swift post container1
Upload	swift upload container1 exercise.sh
Download	swift download container1 exercise.sh
Delete	swift delete container1 exercise.sh
List	swift list swift list container1



© Copyright 2017 Hewlett Packard Enterprise Development LP

23

### CRUD (create, read, update, delete) operations—Four basic functions of persistent storage

Use the Swift CLI client to perform common create, read, update, and delete (CRUD) operations:

- Use `swift post` to create a specified container. You must create a container before you can store objects.
- Use `swift upload` to upload an object in your file system and store it in a specified container.
- Use `swift download` to download a specific object from a specific container. To determine what containers exist and what objects are in a container:
  - Enter `swift list` to list all containers
  - Enter `swift list <container>` to list the contents of a container

## Sample API PUT of an object

- Get a token (account was already created):

```
curl -i -X POST http://10.0.2.15:5000/v2.0/tokens \  
-H "Content-Type: application/json" \  
-d '{"auth": {"tenantName": "admin", \  
"passwordCredentials": {"username": "admin", "password": "xxxxxx"}}}'
```

- Set the value of myToken=<returned token>

- Run swift stat -v to obtain the account ID (AUTH\_xxxx)

- Create a container:

```
curl -i http://10.0.2.15:8080/v1/AUTH_d7cf0245512540b99fb2c917\  
ec3ee3e6/container1 -X PUT -H "X-Auth-Token: $myToken"
```

- Put the object:

```
curl -X PUT -H "x-auth-token: $myToken" -d @test_file \  
https://region-a.geo-1.objects.somecloudsvc.com/v1.0/48666127952678\  
/test_container/object1
```



© Copyright 2017 Hewlett Packard Enterprise Development LP

24

## Sample API PUT of an object

The example content shows the formation of a set of API requests to create a container. Use the cURL commands when you need to see the actual responses returned by the service. If you are performing operations from the command line, you can normally use the Swift Python client, if it has been installed.

To create a container or upload an object, use the `PUT` operation. To update metadata, use a `POST` operation.

**Note:** The paths for your container and object are simply URLs. Those objects could, in fact, be web pages.

## Swift from the Horizon UI

**Listing of containers**

**Drilling down into pseudo folders**

Container	Object Count	Size	Access
testObjectContainer	3	1.4 MB	Private
courseImageFiles	2	1.4 MB	pseudo-folder
M09_Swift_Object_Storage_Jan2016.pptx	1	1.4 MB	object

Object	Object Count	Size	Access
swiftArch.png	1	13.3 KB	object

Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

25

## Swift from the Horizon UI

The Horizon UI provides the ability to create, modify, and add objects to containers, as well as pseudo folders.



## Troubleshooting Swift

## Diagnosing Swift issues

- Swift logs all events in `/var/log/syslog`, so you can first check for errors there
- Make sure all of the server processes are running
  - For base Swift functionality, the proxy, account, container, object, and auth servers should be running
- If one of the servers is not running, and no errors are logged to syslog, it might be useful to try to start the server manually rather than with a script
  - For example, `swift-object-server /etc/swift/object-server/1.conf` starts the object server
- Make sure that the service endpoints for containers, accounts, and related configuration files are correct
- If you see specific problems like `503 Service Unavailable`, double-check the port values to ensure they match the value in `etc/swift/auth-server.conf`
- When you do not run all the Swift servers as root, you can expect to see the following warning: "Unable to increase file descriptor limit. Running as non-root?"
  - These errors are expected and you can safely ignore them



© Copyright 2017 Hewlett Packard Enterprise Development LP

27

### Diagnosing Swift issues

The slide above outlines some common checks to perform while diagnosing Swift issues. For example, get a listing of service endpoints using the Keystone catalog, and then verify your service endpoint.

If you see specific problems like `503 Service Unavailable`, double-check the port values to ensure they match the value in `etc/swift/auth-server.conf`.

When adding a new account or a user, the auth server calls out to the proxy server based on the `default_cluster_url` configuration file value and it should match the `bind_port` value. You also have to configure the port for the `proxy-server.conf` under the `[filter:auth]` section, which defaults to 11000.

## Troubleshooting with Swift log files

- Swift logs to /var/log/syslog
  - Configured during installation
  - When a request is made to Swift, it is given a unique transaction ID that should be in every log line with the request

```

Oct 19 11:17:50 hpdevstack proxy-server Using identity: {'roles': [u'admin'], 'user': u'admin', 'tenant': (u'd7cf0245512540b99fb2c917ec3ee3e6', u'admin')} (txn: txb7a2d0f12fba42b5bc7a6-005262cccd)
Oct 19 11:17:50 hpdevstack proxy-server allow user with role admin as account admin (txn: txb7a2d0f12fba42b5bc7a6-005262cccd) (client_ip: 10.0.2.15)
Oct 19 11:17:50 hpdevstack container-server 127.0.0.1 - - [19/Oct/2013:18:17:50 +0000] "HEAD /sdb1/427/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1" 204 - "txb7a2d0f12fba42b5bc7a6-005262cccd" "HEAD http://10.0.2.15:8080/v1/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1" "proxy-server 31987" 0.0036
Oct 19 11:17:50 hpdevstack object-server 127.0.0.1 - - [19/Oct/2013:18:17:50 +0000] "HEAD /sdb1/385/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1/localrc" 404 - "HEAD http://10.0.2.15:8080/v1/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1/localrc" "txb7a2d0f12fba42b5bc7a6-005262cccd" "proxy-server 31987" 0.0007

```

## Troubleshooting with Swift log files

You can copy a transaction ID from the log file and then grep the log file for all lines containing that ID. For example:

```

$ grep txb7a2d0f12fba42b5bc7a6-005262cccd /var/log/syslog
Oct 19 11:17:49 hpdevstack proxy-server STDOUT: 2013-10-19 18:17:49.430 31987 INFO
urllib3.connectionpool [-] Starting new HTTP connection (1): 10.0.2.15 (txn:
txb7a2d0f12fba42b5bc7a6-005262cccd)

Oct 19 11:17:50 hpdevstack proxy-server Using identity: {'roles': [u'admin'], 'user': u'admin', 'tenant': (u'd7cf0245512540b99fb2c917ec3ee3e6', u'admin')} (txn: txb7a2d0f12fba42b5bc7a6-005262cccd)

Oct 19 11:17:50 hpdevstack proxy-server allow user with role admin as account admin (txn: txb7a2d0f12fba42b5bc7a6-005262cccd) (client_ip: 10.0.2.15)

Oct 19 11:17:50 hpdevstack container-server 127.0.0.1 - - [19/Oct/2013:18:17:50 +0000] "HEAD /sdb1/427/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1" 204 -
"txb7a2d0f12fba42b5bc7a6-005262cccd" "HEAD
http://10.0.2.15:8080/v1/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1" "proxy-
server 31987" 0.0036

Oct 19 11:17:50 hpdevstack object-server 127.0.0.1 - - [19/Oct/2013:18:17:50 +0000] "HEAD /sdb1/385/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1/localrc" 404 - "HEAD
http://10.0.2.15:8080/v1/AUTH_d7cf0245512540b99fb2c917ec3ee3e6/container1/localrc"
"txb7a2d0f12fba42b5bc7a6-005262cccd" "proxy-server 31987" 0.0007

```

## Helpful CLI commands

- Test using a CLI:
  - `swift stat`
- Find an account, a container, or an object:
  - `swift-get-nodes`
- Display object information:
  - `swift-object-info`
- Crawl the account, and check that all containers and objects can be found:
  - `swift-account-audit`



© Copyright 2017 Hewlett Packard Enterprise Development LP

29

## Helpful CLI commands

The following commands can be used to examine the status of the object storage:

- To test the object storage and show statistics, enter `swift-stat`
- To find an account, a container, or an object, enter `swift-get-nodes`
- To display object information, enter `swift-object-info`
- To check all containers and objects connected to a certain account, enter `swift-account-audit`

Example:

```
hp@hpdevstack:~/devstack$ swift stat
Account: AUTH_d7cf0245512540b99fb2c917ec3ee3e6
Containers: 1
Objects: 1
Bytes: 3477
Accept-Ranges: bytes
X-Timestamp: 1382204001.10275
X-Trans-Id: tx239f96314ecd44979cdea-005262e23d
Content-Type: text/plain; charset=utf-8
```

## Module summary

In this module:

- Object storage was defined
- The primary function and features of OpenStack® Swift were discussed
- The features of Swift accounts, containers, and objects were explained
- The Swift architecture and the operation of the Swift ring were explained
- The purpose of Swift zones and regions was explained
- The data consistency features of Swift were explained
- Some common Swift management tasks and troubleshooting methods were presented



© Copyright 2017 Hewlett Packard Enterprise Development LP

30

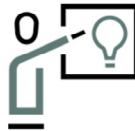
## Module summary

In this module:

- Object storage was defined
- The primary function and features of OpenStack® Swift were discussed
- The features of Swift accounts, containers, and objects were explained
- The Swift architecture and the operation of the Swift ring were explained
- The purpose of Swift zones and regions was explained
- The data consistency features of Swift were explained
- Some common Swift management tasks and troubleshooting methods were presented

# Learning check

## Learning check



Object storage is not generally a good solution for which of the following types of data?

- A. Massive amounts of unstructured data
- B. Data that changes frequently, such as database entries
- C. Storage backups
- D. Video and audio files
- E. Virtual machine images

## Learning check

Object storage is not generally a good solution for which of the following types of data?

- A. Massive amounts of unstructured data
- B. Data that changes frequently, such as database entries
- C. Storage backups
- D. Video and audio files
- E. Virtual machine images

## Learning check answer



Object storage is not generally a good solution for which of the following types of data?

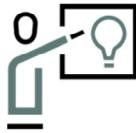
- A. Massive amounts of unstructured data
- B. Data that changes frequently, such as database entries**
- C. Storage backups
- D. Video and audio files
- E. Virtual machine images

## Learning check answer

Object storage is not generally a good solution for which of the following types of data?

- A. Massive amounts of unstructured data
- B. Data that changes frequently, such as database entries**
- C. Storage backups
- D. Video and audio files
- E. Virtual machine images

## Learning check



Which of the following are the features of OpenStack® Swift? (select all the correct responses)

- A. Does not use a conventional file system
- B. Data is distributed evenly throughout the system
- C. Uses a central database
- D. Provides data redundancy (at least 3 replicas)

## Learning check

Which of the following are the features of OpenStack® Swift? (select all the correct responses)

- A. Does not use a conventional file system
- B. Data is distributed evenly throughout the system
- C. Uses a central database
- D. Provides data redundancy (at least 3 replicas)

## Learning check answer



Which of the following are the features of OpenStack® Swift? (select all the correct responses)

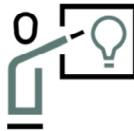
- A. Does not use a conventional file system**
- B. Data is distributed evenly throughout the system**
- C. Uses a central database
- D. Provides data redundancy (at least 3 replicas)**

## Learning check answer

Which of the following are the features of OpenStack® Swift? (select all the correct responses)

- A. Does not use a conventional file system**
- B. Data is distributed evenly throughout the system**
- C. Uses a central database
- D. Provides data redundancy (at least 3 replicas)**

## Learning check



Which of the following Swift services has a database that stores both a container's metadata and a record for each object it contains?

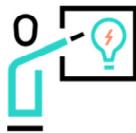
- A. Swift proxy server
- B. Account
- C. Container
- D. Object

## Learning check

Which of the following Swift services has a database that stores both a container's metadata and a record for each object it contains?

- A. Swift proxy server
- B. Account
- C. Container
- D. Object

## Learning check answer



Which of the following Swift services has a database that stores both a container's metadata and a record for each object it contains?

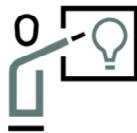
- A. Swift proxy server
- B. Account
- C. Container**
- D. Object

## Learning check answer

Which of the following Swift services has a database that stores both a container's metadata and a record for each object it contains?

- A. Swift proxy server
- B. Account
- C. Container**
- D. Object

## Learning check



What is the purpose of a Swift ring?

- A. Map between logical volumes and physical volumes
- B. Map between external names of stored items and their physical location
- C. Map containers to storage zones
- D. Map an account, a container, or an object to a partition

## Learning check

What is the purpose of a Swift ring?

- A. Map between logical volumes and physical volumes
- B. Map between external names of stored items and their physical location
- C. Map containers to storage zones
- D. Map an account, a container, or an object to a partition

## Learning check answer



What is the purpose of a Swift ring?

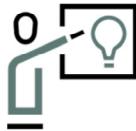
- A. Map between logical volumes and physical volumes
- B. Map between external names of stored items and their physical location
- C. Map containers to storage zones
- D. Map an account, a container, or an object to a partition**

## Learning check answer

What is the purpose of a Swift ring?

- A. Map between logical volumes and physical volumes
- B. Map between external names of stored items and their physical location
- C. Map containers to storage zones
- D. Map an account, a container, or an object to a partition**

## Learning check



What is an attribute of Swift object storage? (Select two)

- A. Scales to zettabytes
- B. Supports multi-tenancy
- C. Provides volumes for instances
- D. Creates file systems for instances
- E. Sizes chunks based on flavors



© Copyright 2017 Hewlett Packard Enterprise Development LP

40

## Learning check

What is an attribute of Swift object storage? (Select two)

- A. Scales to zettabytes
- B. Supports multi-tenancy
- C. Provides volumes for instances
- D. Creates file systems for instances
- E. Sizes chunks based on flavors

## Learning check answer



What is an attribute of Swift object storage? (Select two)

- A. Scales to zettabytes**
- B. Supports multi-tenancy**
- C. Provides volumes for instances
- D. Creates file systems for instances
- E. Sizes chunks based on flavors



© Copyright 2017 Hewlett Packard Enterprise Development LP

41

## Learning check answer

What is an attribute of Swift object storage? (Select two)

- A. Scales to zettabytes**
- B. Supports multi-tenancy**
- C. Provides volumes for instances
- D. Creates file systems for instances
- E. Sizes chunks based on flavors



**Hewlett Packard  
Enterprise**

© Copyright 2017 Hewlett Packard Enterprise Development LP A2

Hewlett Packard  
Enterprise

# Fundamentals of OpenStack® Technology

Module 10—OpenStack® Orchestration Service (Heat)

H6C68S E.01

© Copyright 2017 Hewlett Packard Enterprise Development LP

## Learning objectives

After completing this module, you should be able to:

- Discuss the OpenStack® Heat functionality, components, and commonly used terminology
- Locate the information necessary to install and configure Heat
- Create a Heat template
- Launch a stack with Heat
- Describe basic troubleshooting for the Heat service



© Copyright 2017 Hewlett Packard Enterprise Development LP

2

## Learning objectives

After completing this module, you should be able to:

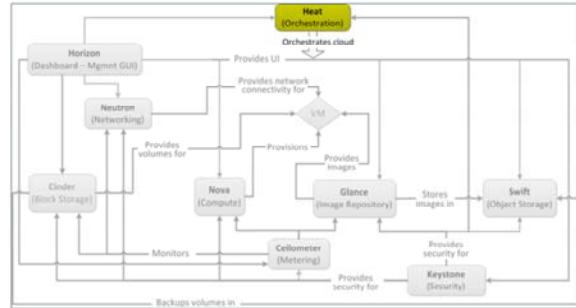
- Discuss the OpenStack® Heat functionality, components, and commonly used terminology
- Locate the information necessary to install and configure Heat
- Create a Heat template
- Launch a stack with Heat
- Describe basic troubleshooting for the Heat service



## Heat overview

## Heat functionality

- Heat:
  - Is a template-based service that executes OpenStack® API calls to generate running cloud applications
  - Provides for the creation of resource types:
    - Instances, Floating IP addresses, volumes, security groups, and so on
    - Advanced services such as high availability, auto-scaling, and nested stacks
  - Integrates with other core OpenStack® components
- All OpenStack® resources and related dependencies are specified in an AWS CloudFormation-compatible template called the Heat Orchestration Template (HOT)
- Management through both an OpenStack®-native REST API and a CloudFormation-compatible Query API



## Heat functionality

Heat is the main project in the OpenStack® Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates that are in the form of text files that can be treated like code. A native Heat template format is evolving, but Heat is also compatible with the Amazon Web Services (AWS) CloudFormation template format, so many existing CloudFormation templates can be launched on OpenStack®.

Heat provides both an OpenStack®-native Representational State Transfer (REST) API and a CloudFormation-compatible Query API.

## Heat terminology

Term	Definition
CFN (Cloud Formation)	<ul style="list-style-type: none"> <li>– Amazon orchestration template format supported by OpenStack® Heat</li> </ul>
HOT (Heat Orchestration Format)	<ul style="list-style-type: none"> <li>– Native orchestration template format for OpenStack® Heat</li> </ul>
IaaS	<ul style="list-style-type: none"> <li>– A virtualized environment that is delivered as a service over the Internet by a provider</li> <li>– The infrastructure can include servers, network equipment, and software</li> </ul>
Template	<ul style="list-style-type: none"> <li>– A declaration of the resources and associated parameters that make up a stack</li> <li>– Strongly typed using a YAML format based template</li> </ul>
Stack	<ul style="list-style-type: none"> <li>– A group of connected cloud resources (VMs, volumes, networks, and so on) that comprise a cloud IaaS architecture application and are created from the submission of a template</li> </ul>
YAML	<ul style="list-style-type: none"> <li>– A format for structuring data in a way that is usable by programming languages, and at the same time readable by humans</li> <li>– Used by the OpenStack® HOT template</li> <li>– YAML is a recursive acronym for "YAML Ain't Markup Language"</li> </ul>



© Copyright 2017 Hewlett Packard Enterprise Development LP

5

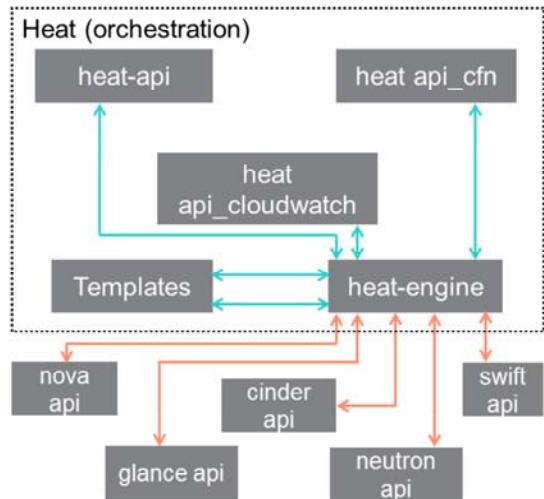
## Heat terminology

The following terms are commonly used in Heat documentation:

- **Infrastructure as a Service (IaaS):** A virtualized environment that is delivered as a service over the Internet by a provider. The infrastructure can include servers, network equipment, and software.
- **Template:** A declaration of the resources and associated parameters that make up a stack. A template can be stored as a text file with a format that complies with the JavaScript Object Notation (JSON) standard. As a plain text file, a template can be edited using any text editor and stored in any source control system.
- **Stack:** A group of connected cloud resources (virtual machines [VMs], volumes, networks, and so on) that comprise a cloud IaaS architecture application and are created from the submission of a template.

## Heat components

- **heat-engine:** Does all the orchestration work and is the layer in which the resource integration is implemented
- **heat-api:** A service that exposes an external REST-based API to the heat-engine service
  - The communication between the heat-api and the heat-engine uses a message queue-based RPC
- **heat-api-cfn:** Provides an AWS Query API that is compatible with the AWS CloudFormation and that processes API requests by sending them to the heat-engine over RPC
- **heat-api-cloudwatch:** A CloudWatch-like API service available to the Heat project



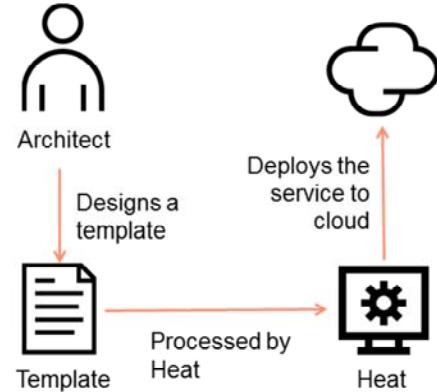
## Heat components

The Heat service consists of four main components:

- **heat engine:** Does all the orchestration work and is the layer in which the resource integration is implemented.
- **heat-api:** A service that exposes an external REST-based API to the heat-engine service. The communication between the heat-api and the heat-engine uses a message queue-based Remote Procedure Call (RPC).
- **heat-api-cfn:** Provides an AWS Query API that is compatible with the AWS CloudFormation and that processes API requests by sending them to the heat-engine over RPC.
- **heat –api-cloudwatch:** A CloudWatch-like API service to the Heat project.

## How a Heat template works

- Describes the infrastructure for a cloud application in a text file that can be committed to the version control repository
- Describes infrastructure resources such as servers, Floating IP addresses, volumes, security groups, and users
- Can be used to specify the relationships between resources
  - For example, which volume is connected to which server
- Specified in the command used to create a stack:  
`heat stack create <stack name> -- template-file =<template name>`
- Can be modified for updating an existing stack
  - Integrates well with software configuration management tools such as Puppet and Chef



## How a Heat template works

A Heat template describes the infrastructure for a cloud application in a text file that is readable and writable by humans, and can be checked into version control, diffed, and so on.

Infrastructure resources that can be described in the template include servers, Floating IPs, volumes, security groups, and users.

Templates can also specify the relationships between resources (for example, this volume is connected to this server). This enables Heat to call out to the OpenStack® APIs to create all of your infrastructure in the correct order to completely launch your application.

Heat manages the whole lifecycle of the application. When you need to change your infrastructure, simply modify the template and use it to update your existing stack. Heat knows how to make the necessary changes. It deletes all of the resources when you are finished with the application.

Heat primarily manages infrastructure, but the templates integrate well with software configuration management tools such as Puppet and Chef. For more information on these two tools, visit <https://s3.amazonaws.com/cloudformation-examples/IntegratingAWSCloudFormationWithPuppet.pdf>.

## HOT (Heat Orchestration Template)

- A template format meant to replace the Heat CFN as the native format supported by Heat
- YAML text file that provides the information necessary to provision an OpenStack® infrastructure
- Can be created or modified from existing AWS or Heat templates
- Supports advanced functionality such as instance high availability, instance auto-scaling, and nested stacks

**NOTE:** HOT support is still under development. More work is required to provide access to all of the functionality currently available through the CFN-compatible template interface.

**NOTE:** The *Heat Orchestration Template (HOT) Guide* is available at: [http://docs.openstack.org/developer/heat/template\\_guide/hot\\_guide.html](http://docs.openstack.org/developer/heat/template_guide/hot_guide.html)



© Copyright 2017 Hewlett Packard Enterprise Development LP

8

```
heat_template_version: 2013-10-15
description: A HOT template that requests the several input and then spins up two instances and a private network.

parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
    default: testkeypair
  image_id:
    type: string
    label: Image ID
    description: Image to be used for compute instance
    default: cirros-0.3.2-x86_64-disk
  instance_flavor:
    type: string
    label: Instance Flavor
    description: Type of instance (flavor) to be used
    constraints:
      - allowed_values: [ m1.tiny, m1.medium, m1.large ]
        description: Value must be one of m1.tiny, m1.medium, m1.large.
  ext_netid:
    type: string
    label: External Network ID
    description: ID of External Network
    default: ""

resources:
  heat_network_01:
    type: OS::Neutron::Net
    properties:
      admin_state_up: true
      name: heat-network-01
  heat_subnet_01:
    type: OS::Neutron::Subnet
    properties:
```

The `heat verify-template` command is designed to check only the syntax of the template. It does not verify if any related properties are valid.

## HOT (Heat Orchestration Template)

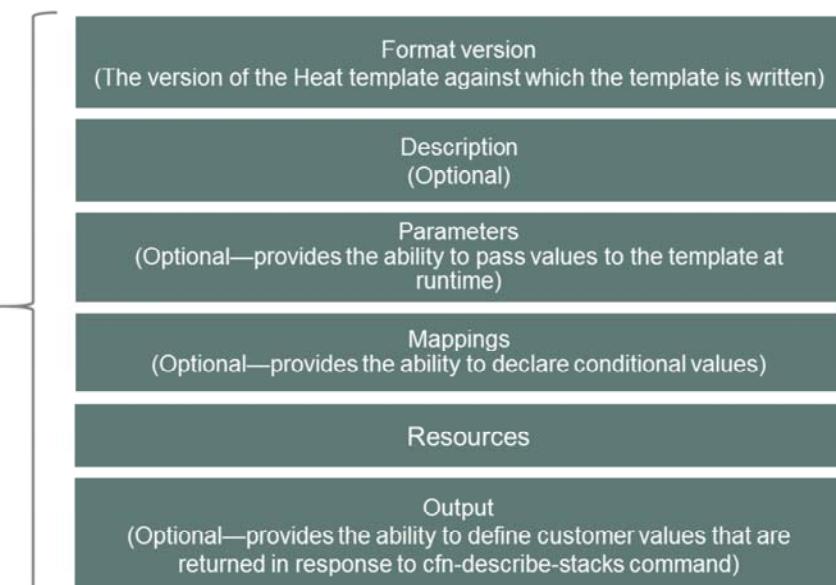
This specification explains in detail all elements of the HOT template format. An example-driven guide to writing HOT templates can be found in *Heat Orchestration Template (HOT) Guide*.

### References:

- <http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html>
- [http://docs.openstack.org/developer/heat/template\\_guide/](http://docs.openstack.org/developer/heat/template_guide/)
- [http://docs.openstack.org/developer/heat/template\\_guide/hot\\_guide.html](http://docs.openstack.org/developer/heat/template_guide/hot_guide.html)

## Heat template components

Template (associated with a stack)



Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

9

### Heat template components

Heat Orchestration Templates (HOT) are defined in YAML:

- **heat\_template\_version:** This key with value 2013-05-23 (or a later date) indicates that the YAML document is a HOT template of the specified version. The version determines the features supported by the template.
- **Description:** Allows for giving a description of the template or the workload that can be deployed using the template.
- **parameter\_groups:** Used to specify how the input parameters should be grouped and the order in which to provide the parameters. This section is optional and can be omitted when necessary.
- **Parameters:** Used to specify the input parameters that have to be provided when instantiating the template. This section is optional and can be omitted when no input is required.
- **Resources:** Used for the declaration of one or more resources to be used to create the stack from the template.
- **Outputs:** Used to specify the output parameters available to users once the template has been instantiated. This section is optional and can be omitted when no output values are required.

## Sample Heat template format (1 of 3)

Format version (required)  
v2015-10-15 supports Newton features

Description

Parameters

```
heat_template_version: 2015-10-15
description: >
  A HOT template that requests the several input and then spins up two
  instances and a private network.

parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
    default: heatkeypair
  image_id:
    type: string
    label: Image ID
    description: Image to be used for compute instance
    default: cirros-0.3.2-x86_64-disk
  instance_flavor:
    type: string
    label: Instance Flavor
    description: Type of instance (flavor) to be used
    constraints:
      - allowed_values: [ ml.tiny, ml.medium, ml.large ]
        description: Value must be one of ml.tiny, ml.medium, ml.large.
  ext_netid:
    type: string
    label: External Network ID
    description: ID of External Network
    default: "
```

Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

10

## Sample Heat template format (1 of 3)

The sample above illustrates the Heat template starting with format version information, followed by description and parameters configuration sections.

## Sample Heat template format (2 of 3)

```

resources:
  wordpress_instance:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_id }
      flavor: { get_param: instance_type }
      key_name: { get_param: key_name }
    user_data:
      str_replace:
        template: |
          #!/bin/bash -v

          yum -y install mariadb mariadb-server httpd wordpress
          touch /var/log/mariadb/mariadb.log
          chown mysql.mysql /var/log/mariadb/mariadb.log
          systemctl start mariadb.service

          # Setup MySQL root password and create a user
          mysqladmin -u root password db_rootpassword
          cat << EOF | mysql -u root --password=db_rootpassword
          CREATE DATABASE db_name;
          GRANT ALL PRIVILEGES ON db_name.* TO "db_user"@"localhost"
          IDENTIFIED BY "db_password";
          FLUSH PRIVILEGES;
          EXIT

```

### Sample Heat template format (2 of 3)

The sample above illustrates the `resources` section of the Heat template. The `resources` section allows you to define parameter references, as well user data to be passed to the instance.

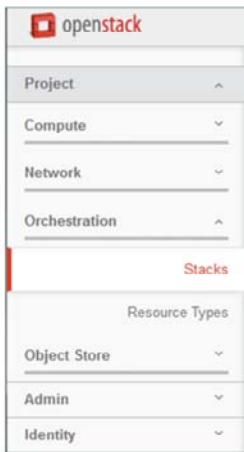
## Example Heat template format (3 of 3)

```
outputs:  
  WebsiteURL:  
    description: URL for Wordpress wiki  
    value:  
      str_replace:  
        template: http://host/wordpress  
        params:  
          host: { get_attr: [apache_floatingip, floating_ip_address] }
```

### Sample Heat template format (3 of 3)

The sample above illustrates the `outputs` section of the Heat template.

## Heat stack



- A collection of OpenStack® resources
- Created through the use of a HOT
  - Can be tracked from the CLI and GUI
  - Automatically optimizes the order of resource creation by dependences
- Available stacks and itemized contents of a stack can be displayed from the CLI and GUI
- The stack event history can be displayed from the CLI and GUI
- The deletion of a stack automatically optimizes the removal of all stack-associated resources

**NOTE:** If for any reason a member resource cannot be created, all of the resources associated with the stack will be deleted, the stack will not be created, and the stack status will be set to CREATE\_FAILED.

## Heat stack

A Heat stack can be defined as a collection of OpenStack® resources. A stack is created through the use of a Heat Orchestration Template. The status of the stack can be tracked from the CLI and GUI.

Orchestration templates are used to start many VMs simultaneously, and Heat can automatically optimize the order of resource creation by dependencies. This is an important feature, considering that there could be dozens of virtual machine templates.

Similarly to the creation of a stack, the deletion of a stack automatically optimizes the removal of all stack-associated resources.

## Orchestration (Heat) API version states as of Newton release

- Orchestration API v1 (**current**)



© Copyright 2017 Hewlett Packard Enterprise Development LP

14

## Orchestration (Heat) API version states as of Newton release

The orchestration API is at the version v1 (**current**).

## Common Heat management tasks

## Obtaining values required for the Heat template

- List available flavors: `openstack flavor list`
- List available images: `openstack image list`
- Check the Heat status: `openstack stack list`
- Display available keys: `openstack keypair list`



© Copyright 2017 Hewlett Packard Enterprise Development LP

16

## Obtaining values required for the Heat template

Values required for the HOT can be obtained using the CLI or GUI. Often, the CLI offers better visibility and structure of data. Depending on the information you want to extract, you can use some of the commands listed on the slide.

## Launching a stack from the command line

– Launch a stack:

```
- heat stack-create --template-file=<template directory/file name> Parameters=<required parameters and/or parameters values to override default> <stack name>
```

– Example:

```
-heat stack-create --template-file hotTemplates/hotTemplate_inputB.yaml --parameters instance_flavor="m1.tiny" stack4labs
```

id	stack_name	stack_status	creation_time	updated_time
ba0ee4cb-6ea8-4a29-9c4d-faf8e5cc23ac	stack4labs	CREATE_IN_PROGRESS	2016-02-15T17:33:12	None

– Verify that the stack was successfully created: `heat stack-list`

id	stack_name	stack_status	creation_time	updated_time
ba0ee4cb-6ea8-4a29-9c4d-faf8e5cc23ac	stack4labs	CREATE_COMPLETE	2016-02-15T17:33:12	None

## Launching a stack from the command line

All of the values specified by the parameters are either used to provide a value for a required parameter as specified by the template or override a value for a parameter specified by the template.

The `stack-list` command provides the ability to verify that the stack was successfully initialized.

If the deployment fails, refer to <https://ask.openstack.org/en/question/9698/devstack-heat-dns-resolution-not-working-from-instance-cloud-init-fails/>.

## Viewing stack details from the CLI

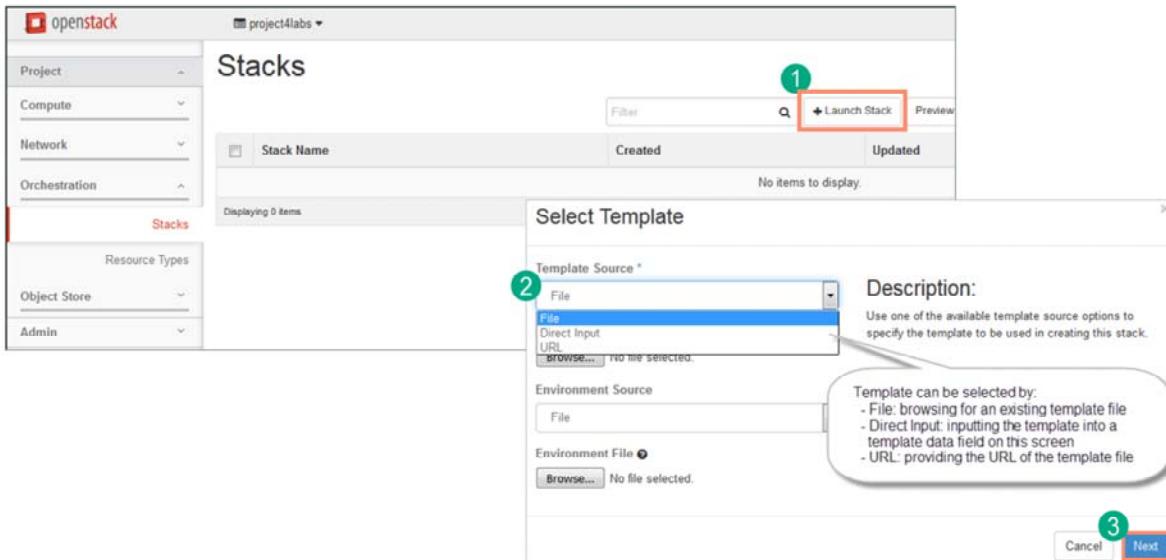
- openstack stack show <stack id or name>

Property	Value
capabilities	[]
creation_time	2016-02-15T17:33:12
description	Accepts provided default values, sets allowed values for inputs, and then deploys a single compute instance
disable_rollback	True
id	ba0ee4cb-6ea8-4a29-9c4d-faf8e5cc23ac
links	[{"http://192.168.117.132:8004/v1/f90855e7beb54db5973db8035d666502/stacks/stack4labs/ba0ee4cb-6ea8-4a29-9c4d-faf8e5cc23ac": "self"}]
notification_topics	[]
outputs	{}
parameters	{"OS::project_id": "f90855e7beb54db5973db8035d666502", "OS::stack_id": "ba0ee4cb-6ea8-4a29-9c4d-faf8e5cc23ac", "OS::stack_name": "stack4labs", "key_name": "heatkeypair", "instance_flavor": "m1.tiny", "image_id": "cirros-0.3.2-x86_64-disk"}
parent	None
stack_name	stack4labs
stack_owner	None
stack_status	CREATE_COMPLETE
stack_status_reason	Stack CREATE completed successfully
stack_user_project_id	38410ee37ac9431a9cf34a2d709b7369
tags	None
template_description	Accepts provided default values, sets allowed values for inputs, and then deploys a single compute instance
timeout_mins	None
updated_time	None

## Viewing stack details from the CLI

The stack details can be displayed using the `heat stack-show` command as well as using the new `openstack stack show` command as illustrated above.

## Launching a stack from the Horizon UI (1 of 3)



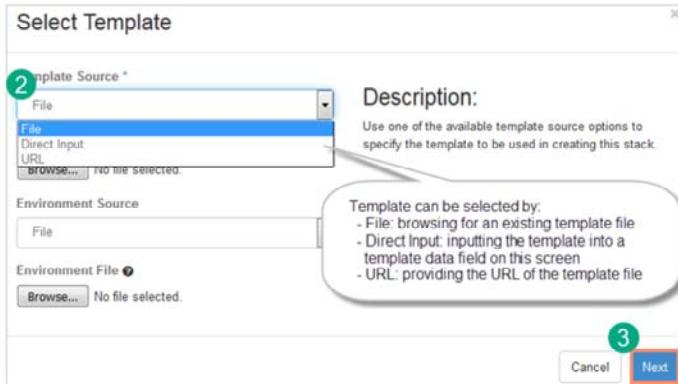
19

### Launching a stack from the Horizon UI (1 of 3)

To begin creating a new stack, log in to Horizon and then:

1. On the left side of the screen, expand the **Project** menu.
2. Click **Orchestration**.
3. Click **Stacks**.
4. On the right side of the screen, click the **Launch Stack** button.

## Launching a stack from the Horizon UI (2 of 3)



- Upload a previously saved template file or directly paste the content of the template to the web GUI
- To paste the template content to the web GUI:
  1. Open the template using any available text editor and copy the content of the file to a clipboard
  2. From the Template Source menu, select **Direct Input**
  3. Paste the content of the template file in the Template Data field and click **Next**

**NOTE:** If you want to upload a template file, select **File** from the **Template Source** menu, and then click **Browse** to select the template file to be uploaded to Horizon.

## Launching a stack from the Horizon UI (2 of 3)

To use a template through Horizon, you have to either upload a previously saved template file or directly paste the content of the template to the web GUI. To paste template content:

1. Open the template using any available text editor and copy the content of the file to a clipboard.
2. Select the **Direct Input** from **Template Source** drop-down list.
3. Paste the content of the template to the Template Data field and click **Next**.

## Launching a stack from the Horizon UI (3 of 3)

Stack Name: stack4labs

Description: Create a new stack with the provided values.

Creation Timeout (minutes): 60

Rollback On Failure:

Password for user "user4labs":  \*\*\*\*\*

**Image ID:** cirros-0.3.2-x86\_64-disk

**Instance Flavor:** m1.tiny

**Key Name:** heatkeypair

Cancel **Launch**

Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

21

4. Provide the information required by the template:

- Stack Name
- Password for the template user (e.g. admin's password)

- Template parameters, which for this template are:

- Image ID
- Instance Flavor
- Key pair name

5. Click **Launch**

### Launching a stack from the Horizon UI (3 of 3)

To launch a stack, you have to enter all of the required information in the launch stack dialog. The required information reflects the parameters defined in the `server:` section of the template you have provided. For example, the screenshot is based on a Heat template with a `server:` section that looks like this:

```
server:
  type: OS::Nova::Server
  properties:
    image: {get_param: image}
    flavor: {get_param: flavor}
    key_name: {get_param: key_name}
    user_data_format: SOFTWARE_CONFIG
```

This means that some of the required parameters have to be defined or created before you can launch a new stack. All of the required information, such as the Universal Unique Identifiers (UUIDs), key names, and so on, can be obtained using either a command line or the GUI.

After you have populated all of the required fields in the dialog, click **Launch** to initiate the new stack creation.

## Viewing Stack Details from the Horizon UI

**Stack Details: stack4labs**

**Stack Overview**

**Information**

Name	stack4labs
ID	61fcfd0-16ab-47af-96a4-d2990b71c42
Description	Accepts provided default values, sets allowed values for inputs, and then deploys a single compute instance

**Status**

Created	2 minutes
Last Updated	Never
Status	Create Complete: Stack CREATE completed successfully

**Outputs**

**Stack Parameters**

**Stack Template**

```

description: Accepts provided default values, sets allowed values for inputs, and then deploys a single compute instance
name: stack4labs
version: '2013-10-15'
parameters:
  image_id: (default: cirros-0.3.2-sd8_64-disk, description: Image to be used for compute instance, label: Image ID, type: string)
  instance_flavor: (default: m1.tiny, description: Flavor of instance (Flavor) to be used, label: Instance Flavor, type: string)
  key_name: (default: heatkeypair, description: Name of key-pair to be used for compute instance, label: Key Name, type: string)
resources:
  stackInputB_Instance:
    properties:
      flavor: (get_param: instance_flavor)
      image_id: (get_param: image_id)
      key_name: (get_param: key_name)
      type: OS::Nova::Server
  
```

**Stack Events**

Stack Resource	Resource	Time Since Event	Status	Status Reason
sd		17 minutes	Create Complete	state changed
sd		17 minutes	Signal In Progress	Signal: deployment succeeded
sd		~ 18 minutes	Create In Progress	state changed
server		18 minutes	Create Complete	state changed
sc		18 minutes	Create Complete	state changed

Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

22

## Viewing Stack Details from the Horizon UI

Depending on the level of complexity in the template and the speed of your hardware, the template creation might take some time. Using Horizon, you can monitor the instance creation initiated by Heat, and you can check the Stack Detail screen for a graphical representation of the template components. During the template creation process, the components that are already configured have a steady green color, and the components that are currently being configured flash in gray.

## Troubleshooting Heat

# Troubleshooting

**Heat/TroubleShooting**

< Heat  
Contents

**Index**

- 1 Instances can't connect to the internet
- 2 OpenStack installation reports error of "unable to write random state"
- 3 nova\_create fails with a timeout error during composition
- 4 I didn't set a parameter correctly in heat and now the template I ran can't be defined
- 5 I get a weird error when using nova\_create
- 6 I got a 302 error when doing a GET
- 7 You get "Quota exceeded: code:InstanceLimitExceeded (HTTP 413)"
- 8 Endpoint not found for heat
- 9 nova spurious errors - Fix your config
- 10 My command query response Keystone not registered
- 11 I added a template and it now doesn't work
- 12 Nova starts creating instances which immediately go to ERROR state
  - It's probably a bug in heat or heatclient
  - If 1.2.2 Mystransfer (COM) before
- 13 Nova update fails with dependency problems related to the "iso" package
- 14 sgpool fails to start
- 15 Openstack daemons can't connect to sgpool
- 16 Libvirt guests can't receive direct assignments from Fedora/RHEL hosts

**Instances can't connect to the internet**

If your instances can't connect to the internet, ensure you have the following nova configuration settings:

```
flat_interface = ens1
public_interface = ens1
```

In this example, the "ens1" interface is being used for an all-in-one openstack install, on Fedora, using the wired interface instead.

It may be necessary to adjust the interface name (e.g to wlan0 or eth0 depending on your OS and network configuration). These config file options are required, or nova won't make the required sgpool rules for the instance when it is created, so it won't be able to access the internet or other network resources.

Also ensure IP forwarding is enabled (make this persistent e.g via /etc/sysctl.conf)

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

24

## Troubleshooting

Troubleshooting problems with the Orchestration service is not an easy task, because Heat provides configuration for the whole stack in a single template file. Therefore, problems might occur at any level.

This section describes some of the common orchestration problems and their solutions:

- Instances cannot connect to the Internet
  - Check if you have enabled IP forwarding on your compute nodes by entering:
 

```
echo 1 > /etc/sys/net/ipv4/ip_forward
```
  - Define the public and private interface names in /etc/nova/nova.conf. Adjust the name of your interface according to your local setup:
 

```
flat_interface = eth0
public_interface = eth0
```
- The OpenStack® installation reports an error of “unable to write random state”
  - If you are executing the openstack script as a non-root user, make sure that ~/rnd is owned by that user:
 

```
stack@hpdevstack:~$ ls -l ~/rnd
-rw----- 1 stack stack 1024 Dec 23 17:09 /opt/stack/.rnd
```

For more information, see the Heat troubleshooting wiki at <https://wiki.openstack.org/wiki/Heat/TroubleShooting>.

## Module summary

In this module:

- The OpenStack® Heat functionality, components, and commonly used terminology were discussed
- The information necessary to install and configure Heat was located
- A Heat template was created
- A stack was launched with Heat
- Basic troubleshooting for the Heat service was described



© Copyright 2017 Hewlett Packard Enterprise Development LP

25

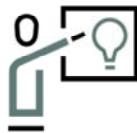
## Module summary

In this module:

- The OpenStack® Heat functionality, components, and commonly used terminology were discussed
- The information necessary to install and configure Heat was located
- A Heat template was created
- A stack was launched with Heat
- Basic troubleshooting for the Heat service was described

# Learning check

## Learning check



What two sections of a HOT template are required?

- A. Resources
- B. Parameters
- C. Format version
- D. Output

## Learning check

What two sections of a HOT template are required?

- A. Resources
- B. Parameters
- C. Format version
- D. Output

## Learning check answer



What two sections of a HOT template are required?

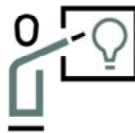
- A. Resources**
- B. Parameters
- C. Format version**
- D. Output

## Learning check answer

What two sections of a HOT template are required?

- A. Resources**
- B. Parameters
- C. Format version**
- D. Output

## Learning check



What is a Heat stack?

- A. A collection of OpenStack® resources
- B. The graphical representation of the orchestration template
- C. The file describing bundled OpenStack® resources



© Copyright 2017 Hewlett Packard Enterprise Development LP

29

## Learning check

What is a Heat stack?

- A. A collection of OpenStack® resources
- B. The graphical representation of the orchestration template
- C. The file describing bundled OpenStack® resources

## Learning check answer



What is a Heat stack?

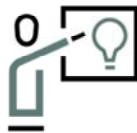
- A. **A collection of OpenStack® resources**
- B. The graphical representation of the orchestration template
- C. The file describing bundled OpenStack® resources

## Learning check answer

What is a Heat stack?

- A. **A collection of OpenStack® resources**
- B. The graphical representation of the orchestration template
- C. The file describing bundled OpenStack® resources

## Learning check



What service is provided by Heat?

- A. SaaS
- B. PaaS
- C. IaaS

## Learning check

What service is provided by Heat?

- A. SaaS
- B. PaaS
- C. IaaS

## Learning check answer



What service is provided by Heat?

- A. SaaS
- B. PaaS
- C. IaaS**

## Learning check answer

What service is provided by Heat?

- A. SaaS
- B. PaaS
- C. IaaS**



**Hewlett Packard  
Enterprise**



**Hewlett Packard  
Enterprise**

# Fundamentals of OpenStack® Technology

Module 11—OpenStack® Telemetry Service (Ceilometer)

H6C68S E.01



## Learning objectives

After completing this module , you should be able to:

- Describe the primary functions of the OpenStack® Ceilometer service
- Describe the high-level architecture of Ceilometer
- Use the Ceilometer CLI command to view data collected by the Ceilometer meters
- View information that Ceilometer collects from the Horizon GUI
- Describe basic troubleshooting for the Ceilometer service



Confidential—For Training Purposes Only

2

## Learning objectives

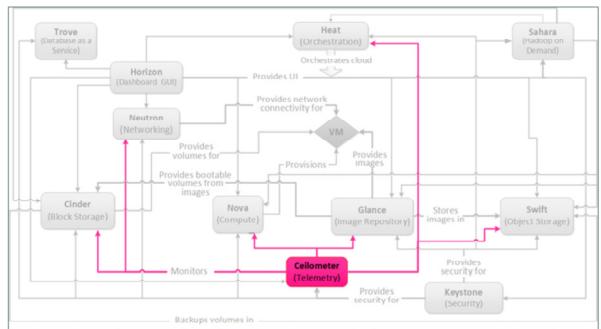
After completing this lab, you should be able to:

- Describe the primary functions of the OpenStack® Ceilometer service
- Describe the high-level architecture of Ceilometer
- Use the Ceilometer CLI command to view data collected by the ceilometer meters
- View information that Ceilometer collects from the Horizon GUI
- Describe basic troubleshooting for the Ceilometer service



## Ceilometer overview

## Ceilometer functionality



– An OpenStack® project for providing metering information about cloud performance and utilization for billing, benchmarking, scalability, and statistics purposes

- Provides efficient collection of metering data
- Can be integrated with the metering system directly or by replacing components
- Collects data through notifications and by polling the infrastructure
- The type of data collected can be configured
- Data is obtained through the REST API
- Metering messages are signed and cannot be repudiated

## Ceilometer functionality

Because OpenStack® provides Infrastructure as a Service (IaaS), it is necessary to be able to meter its performance and utilization for billing, benchmarking, scalability, and statistics purposes. The Ceilometer project provides a single point of contact for billing systems to acquire the measurements needed to establish your billing. This service currently works across all OpenStack® core components.

Ceilometer provides efficient collection of metering data, in terms of processor (CPU) and network costs. It enables those who deploy it to integrate it with the metering system directly or by replacing components, and to configure the type of data collected to meet their operating requirements.

Data can be collected by monitoring the notifications sent from existing services or by polling the infrastructure. The data collected by the metering system is made visible to some users through a Representational State Transfer (REST) API. Metering messages are signed and cannot be repudiated.

## Common Ceilometer use cases

- Providing Cloud Providers with the measurements they need to establish customer billing
- Providing the live metrics necessary to trigger auto-scaling (used by Heat Scaling Groups)
- Publishing information for monitoring, debugging and graphing tools
- Providing the data for data mining and capacity planning



Confidential—For Training Purposes Only

5

## Common Ceilometer use cases

Some of the common ceilometer cases are:

- Providing Cloud Providers with the measurements they need to establish customer billing
- Providing the live metrics necessary to trigger auto-scaling (used by Heat Scaling Groups)
- Publishing information for monitoring, debugging and graphing tools
- Providing the data for data mining and capacity planning

## Ceilometer metering

### Telco industry analogy

#### Metering

- Collecting information about anything that can be billed
- The result of metering is a collection of samples that are ready to be further processed
- A ticket contains the following information on each metered resource:
  - What?
  - Who?
  - When?
  - How much?

#### Rating

- The process of analyzing the series of tickets according to the business rules
- The billable items are attached with the currency value

#### Billing

- The bill line items are assembled into a single, per-customer bill
- The bill might be emitted to start the payment collection

By design, Ceilometer is limited to metering



Confidential – For Training Purposes Only

6

## Ceilometer metering

### Telco industry analogy

To better understand what can be done with the collected data and how the metering process works, this example divides a billing process into three steps that are commonly used in the Telco industry. The three steps are typically defined as follows:

- **Metering:** Collecting information about anything that can be billed
- **Rating:** Analyzing the series of tickets according to the business rules
- **Billing:** Assembling the bill line items into a single, per-customer bill

Ceilometer is limited to the metering function. The decision not to go into rating or billing was made at the beginning, because the variety of private and public clouds seemed too broad for the project to ever deliver a solution that would meet all possibilities. However, you can use Ceilometer data and the Ceilometer API to build your own billing system that does more than metering.

## Ceilometer terminology (1 of 2)

### – Meters:

- Measure a particular aspect of resource usage (e.g. the existence of a running instance) or of ongoing performance (e.g. the current CPU utilization percentage for that instance)
- Exist per-resource (e.g. separate `cpu_util` meter for each instance)
- Continue to exist after the resource has been terminated
- Comprise three values:
  - String name
  - Unit of measurement
  - Type—Cumulative, delta, gauge

**NOTE:** You can find a list of meter types at

<http://docs.openstack.org/developer/ceilometer/measurements.html#user-defined-sample-metadata-for-nova>.



Confidential – For Training Purposes Only

7

## Ceilometer terminology (1 of 2)

Ceilometers meters measure a particular aspect of resource usage (for example, the existence of a running instance) or of ongoing performance (for example, current CPU utilization percentage for that instance). Meters exist per-resource and they continue to exist after the resource has been terminated.

## Ceilometer terminology (2 of 2)

- Samples are individual data points (collection) associated with a particular meter
- Statistics is a set of samples that are aggregated over a time period and that employ 5 different aggregation functions:
  - Count: the number of samples in each period
  - Max: The maximal value of the sample volumes in each period
  - Min: The minimal value of the sample volumes in each period
  - Avg: The average of the sample volumes over each period
  - Sum: The sum of the sample volumes over each period
- Alarms:
  - Are notifications of when a sample crosses a threshold
  - Can be set on a single meter, or on a combination (For example, an alarm can be triggered when memory consumption reaches 70% on a given instance, if it has been up for more than 10 minutes)
  - Can be sent to a log or POST to an HTTP URL for actions

**NOTE:** The alarm function is part of the Aodh project.



Confidential – For Training Purposes Only

8

## Ceilometer terminology (2 of 2)

Ceilometer samples are individual data points (collection) associated with a particular meter. A set of samples aggregated over a certain period of time is called “statistics.” Statistical information is aggregated by the maximal and minimal values, count, averages, and sums. Ceilometer alarms are triggered when a sample crosses a threshold. As of Liberty release (and in Newton), the alarm function is part of the Aodh project.

## Telemetry (Ceilometer) API version states as of Newton release

- Telemetry API v2 (**current**)



Confidential—For Training Purposes Only

9

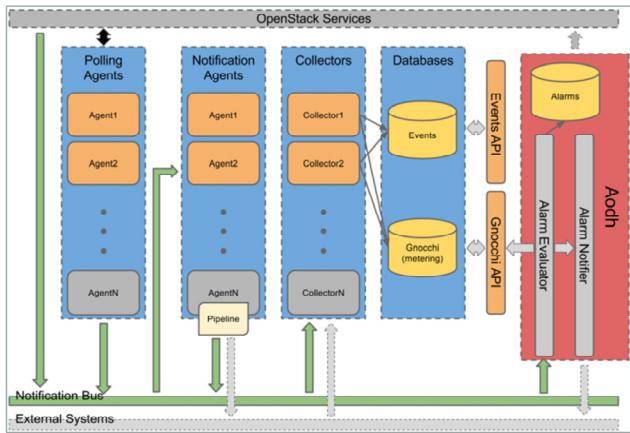
## Telemetry (Ceilometer) API version states as of Newton release

In Newton release, Telemetry API is at the version 2.



## Ceilometer architecture

## Ceilometer architectural components



**NOTE:** The diagram above is taken from <http://docs.openstack.org/developer/ceilometer/architecture.html>

Hewlett Packard  
Enterprise

Confidential – For Training Purposes Only

11

- Each Ceilometer service is designed to scale horizontally
- **Polling agents** poll systems and publish metering messages
- **Notification agents** is a daemon designed to listen to notifications on message queue, convert them to events and samples, and apply pipeline actions
- **Collectors** read metering messages and send data through dispatchers
- **Events API** queries and views data recorded by the collector service DBs
- **Gnocchi** is an OpenStack® project developed to capture metering data in a time series database
- **Aodh** is OpenStack® project designed to manage alarms and notifications

## Ceilometer architectural components

Ceilometer collects data from core OpenStack® projects to produce metering data (samples). Almost every part of Ceilometer is designed to support plug-in extension. The polling agent plug-ins poll services and generate metering data. The notification agent plug-ins receive notification messages and use the notifications to generate metering data.

Metering data is emitted through queues and, optionally, to external services. The Collector supports the collector plug-ins which receive metering data and emit it to the collector dispatchers. The Collector supports multiple dispatchers of various types, including databases (at least one of which typically updates the main Ceilometer database), files in the filesystem, and HTTP endpoints. Plug-in polling agents, notification agents, collectors and dispatchers can be created easily and added to a running Ceilometer system.

Polling and notification pipelines allow transformer plug-ins to modify metering data. Multiple transformers can be connected in a pipeline. Pipelines end with dispatcher plug-ins which emit metering data at the end of the pipeline. Multiple dispatchers can be connected to send sample data to various locations (RabbitMQ, UDP Multicast, and so on).

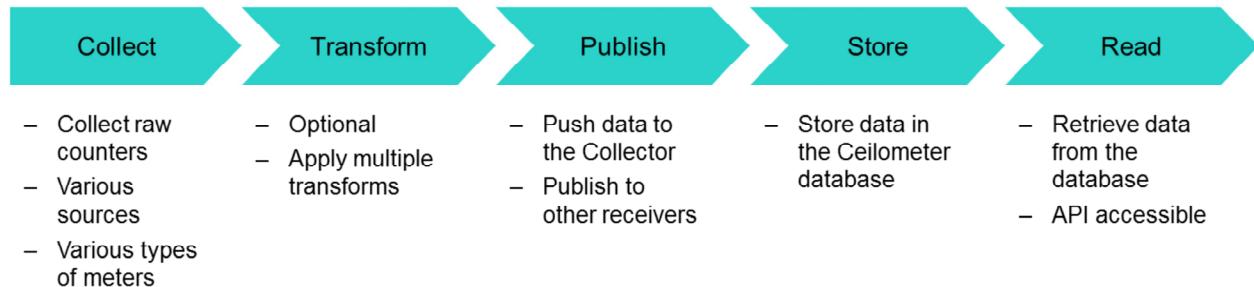
Ceilometer consists of the following components:

- **ceilometer-acentral**: Runs on one or more central management servers and loads polling agent plug-ins to poll for resource utilization and emit metering data
- **ceilometer-acompute**: Polling agent that runs on each compute node and polls for compute instance resource utilization statistics
- **ceilometer-aipmi**: Intelligent Platform Management Interface, for board sensors and Intel node data
- **ceilometer-anotification**: Runs on one or more central management servers, loads notification agent plug-ins to read notifications and emit metering data

- **ceilometer-collector**: Runs on one or more central management servers, loads collector plug-ins to read metering data, and emits data to dispatcher plug-ins (database writers, file writers, HTTP writers, and so on)
- **ceilometer-api**: REST API server that runs on one or more central management servers to provide access to the data from the Ceilometer database
- **aodh**: Aodh alarm service (for Ceilometer)
- **aodh-api**: Aodh REST API (runs under Apache)
- **aodh-notifier**: Aodh/Ceilometer alarm notifier
- **aodh-evaluator**: Aodh/Ceilometer alarm evaluator
- **aodh-listener**: Aodh listener

## Processing the Ceilometer data

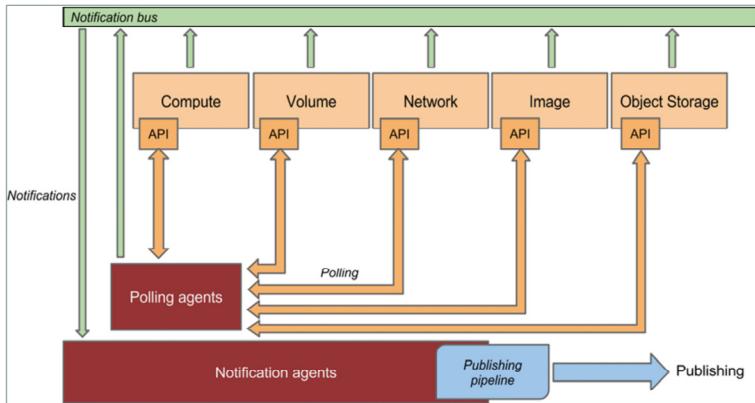
- The process flow for gathering and processing the Ceilometer data is referred to as the “Ceilometer pipeline”
- The pipeline is a set of transformers that manipulate the sample data into something publishers know how to send to external systems



## Processing the Ceilometer data

Ceilometer employs a workflow that collects, transforms, publishes, and stores data in a database. The process flow for gathering and processing the Ceilometer data is referred to as the “Ceilometer pipeline.” The pipeline is a set of transformers that manipulate the sample data into something publishers know how to send to external systems.

## Ceilometer data collection



**NOTE:** The diagram above is taken from  
<http://docs.openstack.org/developer/ceilometer/architecture.html>

- The Ceilometer project uses three methods to collect data:
  - **Notification agents** use the bus listener agent to take events generated on the notification bus and transform them into Ceilometer samples (**the preferred method** of data collection)
  - **Polling agents** poll some API or other tool to collect information at a regular interval (not the preferred option due to the load it can impose on the API services)
  - **Push agents** are added to each node to be monitored and used to push data to the collector (not shown in the diagram)

Hewlett Packard  
Enterprise

Confidential – For Training Purposes Only

14

### Ceilometer data collection

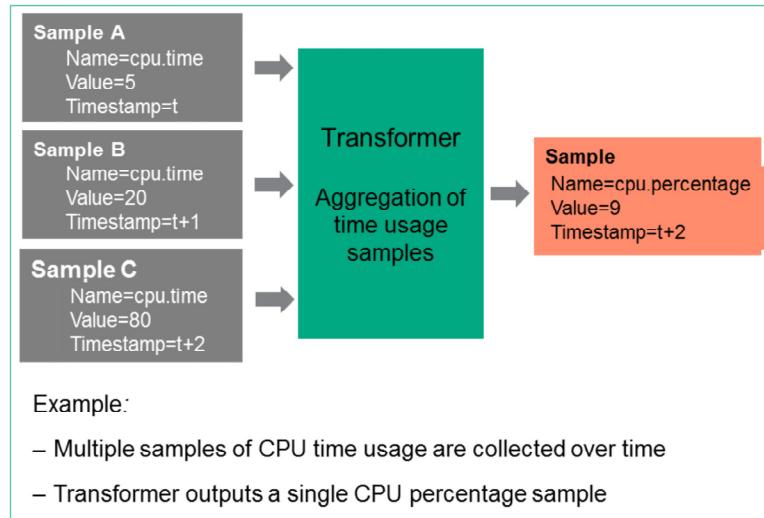
OpenStack® maintains a common notification bus (called Oslo). Each OpenStack® project can publish data on that bus. The Ceilometer Collector contains a listening agent that gathers data messages from the bus.

In theory, every project would send events on the Oslo bus about any metric of interest; however, not all projects have implemented this functionality, so you must instrument other tools that do not use the same bus. To overcome this, Ceilometer supports three independent data collection methods:

- **Bus listener agent**—Takes events generated on the Oslo notification bus and transforms them into Ceilometer samples. This is the preferred method of data collection.
- **Polling agents**—Poll an API or other tool to collect information at a regular intervals. The main drawback is that this method is not resilient.
- **Push agents**—Fetch data from projects that do not otherwise expose the required data in a remotely useable way. This process adds complexity by requiring a component to be added to each of the nodes to be monitored. This method is preferred to polling agents because it is more resilient.

The first method is supported by the ceilometer-collector agent, which monitors the message queues for notifications and for metering data coming from the push and polling agents. The second and third methods rely on a combination of the ceilometer-central-agent, the ceilometer-compute-agent, and the Collector.

## Transforming the Ceilometer data (optional)



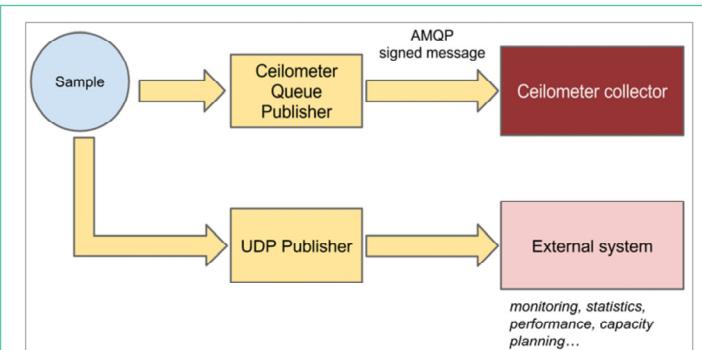
- Transformers are used to gather data samples, manipulate them, and output the results into a form useful for some downstream consumer
- Polling and notifications agents gather a rich set of data, which, if combined with historical or temporal context, can be used to derive even more data
- Ceilometer offers various transformers which can be used to manipulate data in the pipeline
- Output from a transformer can be fed into other transformers

## Transforming the Ceilometer data (optional)

Transformers are used to gather data samples, manipulate them, and output the results into a form useful for some downstream consumer. The polling and notifications agents gather a rich set of data which, if combined with historical or temporal context, can be used to derive even more data.

Ceilometer offers various transformers which can be used to manipulate data in the pipeline. The output from a transformer can be fed into other transformers.

## Publishing the Ceilometer data



- The above example shows the processed Ceilometer data being published to multiple destinations

**NOTE:** The diagram above is taken from  
<http://docs.openstack.org/developer/ceilometer/architecture.html>

Hewlett Packard  
Enterprise

Confidential – For Training Purposes Only

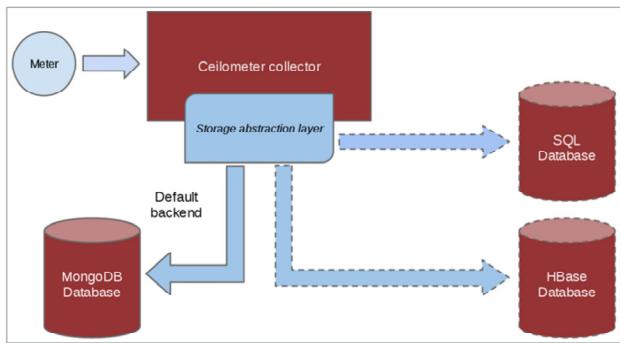
16

## Publishing the Ceilometer data

The processed data can be published using several different transports:

- **Notifier**, a notification-based publisher which pushes samples to a message queue which can be consumed by the collector or an external system
- **UDP**, which publishes samples using UDP packets
- **Kafka**, which publishes data to a Kafka message queue to be consumed by any system that supports Kafka

## Storing the Ceilometer data



- The above example shows the processed Ceilometer data being published to multiple destinations

**NOTE:** The diagram above is taken from  
<http://docs.openstack.org/developer/ceilometer/architecture.html>

Hewlett Packard  
Enterprise

Confidential – For Training Purposes Only

17

- Ceilometer uses a plug-in model to allow data to be stored to various types of database backends
- The Ceilometer database is divided into three separate connections:
  - Alarms
  - Events
  - Metering
- This allows a deployer to store all data within a single database or to divide the data into their own databases
- Ceilometer's storage service is designed to handle use cases where full fidelity of the data is required (e.g. auditing)

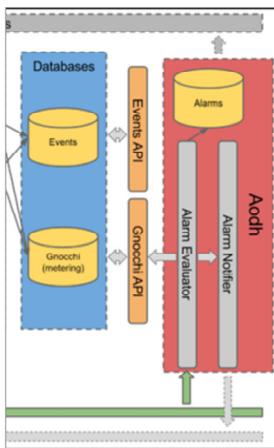
## Storing the Ceilometer data

Ceilometer uses a plug-in model to allow data to be stored to various types of database backends. The Ceilometer database is divided into three separate connections:

- Alarms
- Events
- Metering

This allows a deployer to store all data within a single database or to divide data into their own databases. The Ceilometer storage service is designed to handle use cases where full fidelity of data is required (for example, auditing).

## Ceilometer alarms (Aodh)



- Alarms enable Ceilometer to notify a receiver when a meter achieves a specified threshold value
- Setting alarms:
  - Set on a single meter or on a collection of meters
  - Set alarms from within instances (application alarms)
  - Use the CLI or API
- Notifications are triggered by reaching a sample value threshold
- Alarm actions:
  - Callback URL with parameters
  - Logging
- As of Liberty, alarming support (and subsequently its database) is handled by the OpenStack® Aodh project

## Ceilometer alarms (Aodh)

The alarm component of Ceilometer (first delivered in the Havana release) enables you to set alarms based on a threshold evaluation for a collection of samples. An alarm can be set on a single meter or on a combination of meters.

**Example:** You want an alarm to be triggered when the memory consumption reaches 70% on a given instance, if the instance has been up for more than 10 minutes. To set up this alarm, you would call the Ceilometer API server, specifying the alarm conditions and the action to take.

As of Liberty, alarming support, and subsequently its database, is handled by Aodh.



## Common Ceilometer management tasks

## Common CLI tasks (1 of 2)

List available meters: `ceilometer meter-list`

Name	Type	Unit	Resource ID	User ID	Project ID
cpu	cumulative	ns	5b08da14-782b-41fa-a422-136df24efb6	8b2f46517ce7407d828d98f63c87c0a6	f9085e7beb54db5973db8035d666502
cpu	cumulative	ns	8587270b-b323-4135-846b-1fd60680e74	8b2f46517ce7407d828d98f63c87c0a6	f9085e7beb54db5973db8035d666502
cpu	cumulative	ns	b4edc3b8-25f6-436f-a220-049cd57daeab	8b2f46517ce7407d828d98f03c87c0a6	f9085e7beb54db5973db8035d666502

List available resources: `ceilometer resource-list`

Resource ID	Source	User ID	Project ID
01ae2928666a470b80cedcdebe8916b	openstack	ecb9a5f7eb194391b284ea3e4ed28117	01ae2928666a470b80cedcdebe8916b
01ae2928666a470b80cedcdebe8916b/glance	None		01ae2928666a470b80cedcdebe8916b
06ea4de0a52d42d08343e8e0ce821690	openstack	ecb9a5f7eb194391b284ea3e4ed28117	01ae2928666a470b80cedcdebe8916b

Obtain a meter value, for example `cpu: ceilometer statistics -m cpu`

Period	Period Start	Period End	Max	Min	Avg	Sum	Count	Duration
0	2016-02-12T21:03:12.094055	2016-02-16T02:03:12.635887	7.0956e+11	190000000.0	3.18726227608e+11	3.0247119e+14	949	277200.541832

Duration Start	Duration End
2016-02-12T21:03:12.094055	2016-02-16T02:03:12.635887

Hewlett Packard  
Enterprise

Confidential—For Training Purposes Only

20

## Common tasks (1 of 2)

To list available Telemetry meters, enter `ceilometer meter-list`.

To list available resources, enter `ceilometer resource-list`.

To obtain a meter value, enter `ceilometer statistics -m cpu`.

## Common tasks (2 of 2)

### Use the query option to obtain specific details

```
- ceilometer meter-list --query "project_id=<PROJECT_UUID>"
```

### Set the telemetry alarm

```
- ceilometer alarm-threshold-create --name <NAME> --meter-name <METRIC> --threshold <THRESHOLD> [-q <QUERY>]
```

### Commit the user-defined data

```
- curl -X POST -H 'X-Auth-Token: $myToken' -H 'Content-Type: application/json' -d '<your_sample_list>' \ http://localhost:8777/v2/meters/<custom_meter_name>
```

## Common tasks (2 of 2)

To obtain details about a specific project, enter:

```
ceilometer meter-list --query "project_id=<PROJECT_UUID>"
```

To set the telemetry alarm, enter:

```
ceilometer alarm-threshold-create --name <NAME> --meter-name <METRIC> --threshold <THRESHOLD> [-q <QUERY>]
```

To commit user-defined data, enter:

```
curl -X POST -H 'X-Auth-Token:$myToken' -H 'Content-Type: application/json'\ -d '<your_sample_list>' http://localhost:8777/v2/meters/<custom_meter_name>
```

## Ceilometer info in the Horizon UI

The screenshot shows the OpenStack Horizon interface with the 'Resource Usage' tab selected under the 'System' menu. On the left, a sidebar lists various project and system resources. The main area displays a 'Resources Usage Overview' table with data from the 'Usage Report' tab. The table includes columns for Project, Service, Meter, Description, Day, Value (Avg), and Unit. A modal window titled 'Resources Usage Overview' is open, allowing users to filter metrics by project, group by project, and choose aggregation levels (Avg, Min, Max, Sum). The 'From' and 'To' fields show a date range from March 15, 2016, to March 16, 2016. A legend on the right lists various Ceilometer metrics categorized by service: Compute (Nova), Image (Glance), and Object Storage (Swift).

Project	Service	Meter	Description	Day	Value (Avg)	Unit
admin	Glance	image.size	Uploaded image size	2016-03-16	56,283,325.4	B
project4labs	Glance	image.size	Uploaded image size	2016-03-16	16,299,776.0	B
project4labs	Glance	image.serve	Image is served out			
project4labs	Glance	image.download	Image is downloaded			
admin	Glance	image	Image existence check			
project4labs	Swift_meters	storage.objects.containers	Number of containers			
service	Swift_meters	storage.objects.containers	Number of containers			
admin	Swift_meters	storage.objects.containers	Number of containers			
demo	Swift_meters	storage.objects.containers	Number of containers			
swiftenantest2	Swift_meters	storage.objects.containers	Number of containers			
swiftenantest1	Swift_meters	storage.objects.containers	Number of containers			
project4labs	Swift_meters	storage.objects.containers	Number of containers			

Hewlett Packard  
Enterprise

Confidential—For Training Purposes Only

22

## Ceilometer info in the Horizon UI

There are currently no Ceilometer settings that can be modified in the Horizon UI, but you can view information obtained by Ceilometer from the **Admin → System → Resource Usage** option. When clicked, the option displays the Resources Usage Overview screen that has two tabs to display the metered information, as shown in the above screenshot.



## Ceilometer troubleshooting

## Ceilometer logging

- Monitor the Ceilometer service using the logs in `/var/log/ceilometer-api`
- Set the logging level in the Ceilometer configuration file
  - `verbose = True`
  - `debug = True`



Confidential—For Training Purposes Only

24

### Ceilometer logging

When the Ceilometer service is up and running, it can be monitored using the logs available at `/var/log/ceilometer-api`. The verbosity of the log output can be increased by using the `verbose` and `debug` options in Ceilometer configuration file.

## Module summary

In this module:

- The primary functions of the OpenStack® Ceilometer service were described
- The high-level architecture of Ceilometer was described
- The Ceilometer CLI command was used to view data collected by the Ceilometer meters
- Information that Ceilometer collected from the Horizon GUI was viewed
- Basic troubleshooting for the Ceilometer service was described



Confidential—For Training Purposes Only

25

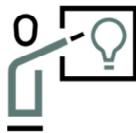
## Module summary

In this module:

- The primary functions of the OpenStack® Ceilometer service were described
- The high-level architecture of Ceilometer was described
- The Ceilometer CLI command was used to view data collected by the Ceilometer meters
- Information that Ceilometer collected from the Horizon GUI was viewed
- Basic troubleshooting for the Ceilometer service was described

# Learning check

## Learning check



To what category does the functionality of collecting samples (also known as “tickets”) belong?

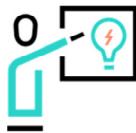
- A. Metering
- B. Rating
- C. Billing

## Learning check

To what category does the functionality of collecting samples (also known as “tickets”) belong?

- A. Metering
- B. Rating
- C. Billing

## Learning check answer



To what category does the functionality of collecting samples (also known as “tickets”) belong?

- A. Metering
- B. Rating
- C. Billing



Confidential—For Training Purposes Only

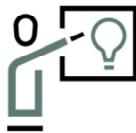
28

## Learning check answer

To what category does the functionality of collecting samples (also known as “tickets”) belong?

- A. Metering
- B. Rating
- C. Billing

## Learning check



What collection mode requires a component to be added to each node to be monitored?

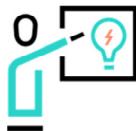
- A. Collection via notifications
- B. Collection via pooling agents
- C. Collection via push agents

## Learning check

What collection mode requires a component to be added to each node to be monitored?

- A. Collection via notifications
- B. Collection via pooling agents
- C. Collection via push agents

## Learning check answer



What collection mode requires a component to be added to each node to be monitored?

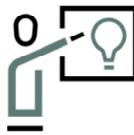
- A. Collection via notifications
- B. Collection via pooling agents
- C. Collection via push agents**

## Learning check answer

What collection mode requires a component to be added to each node to be monitored?

- A. Collection via notifications
- B. Collection via pooling agents
- C. Collection via push agents**

## Learning check



Which new OpenStack® service now handles the alarm functions of Ceilometer?

- A. Gnocci
- B. Ironic
- C. Aodh
- D. Sahara

## Learning check

Which new OpenStack® service now handles the alarm functions of Ceilometer?

- A. Gnocci
- B. Ironic
- C. Aodh
- D. Sahara

## Learning check answer



Which new OpenStack® service now handles the alarm functions of Ceilometer?

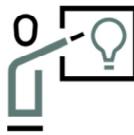
- A. Gnocci
- B. Ironic
- C. Aodh**
- D. Sahara

## Learning check answer

Which new OpenStack® service now handles the alarm functions of Ceilometer?

- A. Gnocci
- B. Ironic
- C. Aodh**
- D. Sahara

## Learning check



Which new OpenStack® service now handles the time series database functions of Ceilometer?

- A. Gnocci
- B. Ironic
- C. Aodh
- D. Sahara

## Learning check

Which new OpenStack® service now handles the time series database functions of Ceilometer?

- A. Gnocci
- B. Ironic
- C. Aodh
- D. Sahara

## Learning check answer



Which new OpenStack® service now handles the time series database functions of Ceilometer?

- A. Gnocci
- B. Ironic
- C. Aodh
- D. Sahara

## Learning check answer

Which new OpenStack® service now handles the time series database functions of Ceilometer?

- A. **Gnocci**
- B. Ironic
- C. Aodh
- D. Sahara



**Hewlett Packard  
Enterprise**



**Hewlett Packard  
Enterprise**

# Fundamentals of OpenStack® Technology

Module 12—Other OpenStack® Projects

H6C68S E.01

© Copyright 2017 Hewlett Packard Enterprise Development LP

## Learning objectives

After completing this module, you should be able to describe:

- The Charging and Billing service (CloudKitty)
- The Key Management service (Barbican)
- The DNSaaS component (Designate)
- Freezer (Backup as a Service)
- Fuel (OpenStack® Environment Provisioning)
- The functionality of OpenStack® Bare Metal Provisioning (Ironic)
- Magnum (Container Virtualization)
- The Shared File System service (Manila)
- The Data Processing service (Sahara)
- The Database as a Service (Trove)
- The Multi-tenant Cloud Messaging service (Zaqar)



© Copyright 2017 Hewlett Packard Enterprise Development LP

2

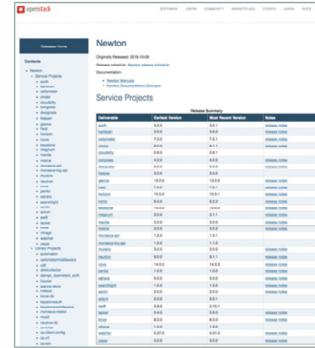
## Learning objectives

After completing this module, you should be able to describe:

- The Charging and Billing service (CloudKitty)
- The Key Management service (Barbican)
- The DNSaaS component(Designate)
- Freezer (Backup as a Service)
- Fuel (OpenStack® Environment Provisioning)
- The functionality of OpenStack® Bare Metal Provisioning (Ironic)
- The Shared File System service (Manila)
- The Data Processing service (Sahara)
- The Database as a Service (Trove)
- The Multi-tenant Cloud Messaging service (Zaqar)

# **OpenStack® service course coverage**

Core services covered in this course	Non-core services covered in this course	Non-core services covered in this module	Other services
<ul style="list-style-type: none"><li>- Cinder</li><li>- Glance</li><li>- Keystone</li><li>- Neutron</li><li>- Nova</li><li>- Swift</li></ul>	<ul style="list-style-type: none"><li>- Cellometer</li><li>- Heat</li><li>- Horizon</li></ul>	<ul style="list-style-type: none"><li>- CloudKitty</li><li>- Barbican</li><li>- Designate</li><li>- Freezer</li><li>- Ironic</li><li>- Manila</li><li>- Sahara</li><li>- Trove</li><li>- Zaqar</li></ul>	<ul style="list-style-type: none"><li>- Chef</li><li>- Congress</li><li>- Cue</li><li>- Fuel</li><li>- Magnum</li><li>- Mistral</li><li>- Monasca</li><li>- Ansible</li><li>- Puppet</li><li>- Rally</li><li>- Sanlin</li><li>- Solum</li><li>- Winstackers</li></ul>



**NOTE:** The most current list of OpenStack® projects and links to those projects are located at <https://releases.openstack.org/newton/index.html#newton-cloudkitty>.

**Hewlett Packard  
Enterprise**

© Copyright 2017 Hewlett Packard Enterprise Development LP

3

# **OpenStack® service course coverage**

The table above lists some of the OpenStack® services. For the full list, please refer to <https://releases.openstack.org/newton/index.html#newton-cloudkitty>.



## CloudKitty (Chargeback and Billing)

## The state of chargeback and billing in OpenStack®

- The use case for CloudKitty:
  - Ceilometer provides telemetry, but billing is not in the scope
  - Commercial tools are often expensive and complex
  - Cloud providers developed internal (custom) solutions
- Billing is fully integrated inside Horizon



© Copyright 2017 Hewlett Packard Enterprise Development LP

5

## The state of chargeback and billing in OpenStack®

CloudKitty is an OpenStack® project that implements chargeback and billing based on the Ceilometer telemetry data. CloudKitty is built to address the following issues:

- Cloud providers have developed custom solutions for billing that are not applicable to OpenStack®.
- Commercial tools are often based on the complex infrastructure, and expensive to implement.

The user interface for chargeback and billing is fully integrated with Horizon.

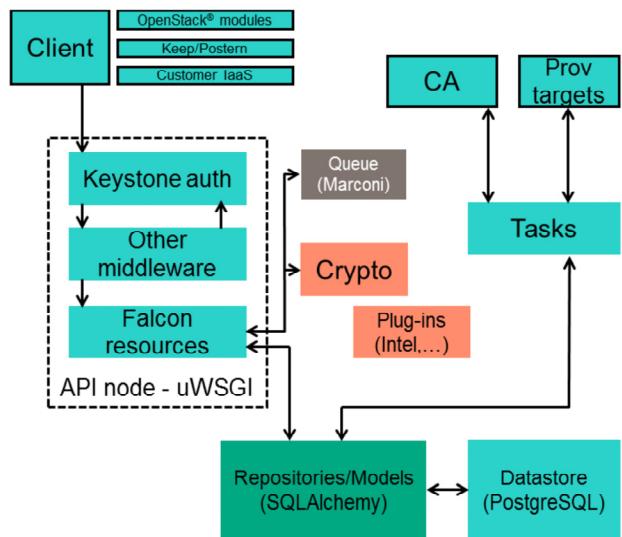


## Barbican (Key Management)

## What is Barbican?

- A REST API designed for the secure storage, provisioning, and management of secrets
  - It is intended for use in all environments, including large ephemeral clouds

**NOTE:** See <https://wiki.openstack.org/wiki/Barbican> for links to the current project development status.



 Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

7

## What is Barbican?

Barbican is a REST API designed for the secure storage, provisioning, and management of secrets. It is aimed at being useful for all environments, including large ephemeral clouds.



## Designate (DNS as a Service)

## What is Designate?

- Designate provides DNSaaS services for OpenStack®
  - REST API for domain and record management
  - Multi-project
  - Integrated with Keystone for authentication
  - A framework is in place to integrate with Nova and Neutron notifications (for auto-generated records)
  - Support for PowerDNS and Bind9 out of the box

**NOTE:** See <https://wiki.openstack.org/wiki/Designate> for links to the current project development status.



© Copyright 2017 Hewlett Packard Enterprise Development LP

9

## What is Designate?

Designate is a recently incubated OpenStack® project that is intended to solve the DNS needs of OpenStack® by providing a RESTful API for managing DNS data with a variety of databases on many different backends. Additionally, Designate provides a service for integrating with Nova and Neutron.

Designate is designed to provide several DNS-as-a-Service (DNSaaS) services for OpenStack®:

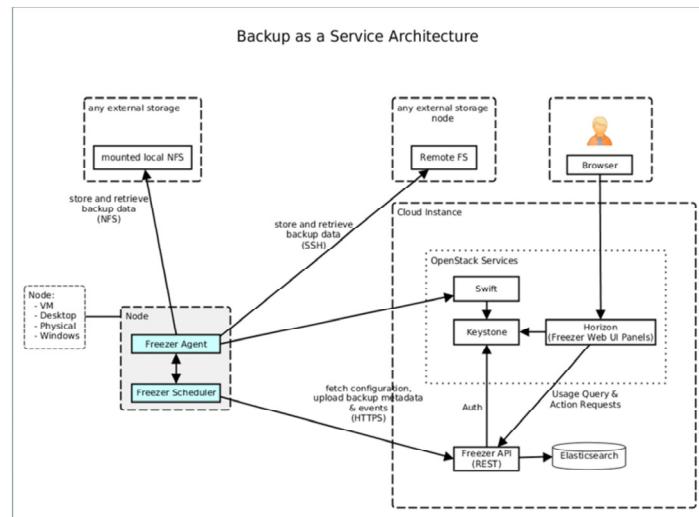
- REST API for domain and record management
- Multi-tenancy
- Integration with Keystone for authentication
- A framework to integrate with Nova and Neutron notifications (for auto-generated records)
- Support for PowerDNS and Bind9 out of the box



## Freezer (Backup as a Service)

## Backup as a Service architecture

- Freezer client components:
  - Freezer agent to perform the backup operation
  - Freezer scheduler to orchestrate a backup
- Freezer service components:
  - Freezer UI
  - Freezer API
- OpenStack® Swift for storage
- GNU Tar to execute backup and restore



Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

11

### Backup as a Service architecture

Freezer has four different components: Freezer Web UI, Freezer scheduler, Freezer agent, and Freezer API. Freezer scheduler and Freezer agent are installed on the client side and Freezer web UI, Freezer API on OpenStack® controller server (The server where your Horizon and Keystone are installed).

Freezer agent is the CLI client that is used to commit jobs to the backup queue. The backup queue is managed by the backup scheduler. To offload the cloud controller server, both agent and scheduler processes are running on the compute node. The Freezer API component is running at the controller plane, to receive communication from agent and scheduler.

## Backup service features

- Backup targets:
  - Local folder
  - SSH server
  - OpenStack® Swift
  - Nova backups (using snapshots)
  - Cinder backups (using cinder backup mechanism)
- Uses GNU Tar for:
  - Incremental backup
  - Full backup
- Designed to reduce to the minimum I/O, CPU, and memory usage



© Copyright 2017 Hewlett Packard Enterprise Development LP

12

## Backup service features

Freezer supports five types of backup targets (backup destinations to send data):

- Local folder—The directory on the local disk of the backup node
- SSH server—Remote SSH server which accepts data over the SSH protocol, and stores it on its local disk
- OpenStack® Swift storage—Freezer API sends the backup stream over the network to be stored as Swift object
- Nova—if you provide Nova arguments in the parameters, Freezer assumes that all necessary data is located on the instance disk and it can be successfully stored using Nova snapshot mechanism
- Cinder backups—Cinder has its own mechanism for backups, and Freezer supports it, but it also allows creating a Glance image from a volume and uploading to Swift

Freezer is designed to reduce to the minimum I/O, CPU, and memory usage. This is achieved by generating a data stream from tar (for archiving) and gzip (for compressing). Freezer segments the stream in a configurable chunk size (with the option `--max-seg-size`). The default segment size is 64MB, so it can be safely stored in memory, encrypted if the key is provided, and uploaded to Swift as a segment.

## Example usage

- To start the backup job:
  - Create the JSON file (example on the right-hand side)
  - Upload the job into the API:
 

```
freezer-scheduler -c node12 job-create
--file test_job.json
```
  - List the newly created job:
 

```
freezer-scheduler -c node12 job-list
```
  - Display the content of the job:
 

```
freezer-scheduler -c node12 job-get -j <job_id>
```
  - Start the scheduler on the target node (commit the job):
 

```
freezer-scheduler -c node12 -i 15 -f ~/job_dir start
```
- More information available at :
 <https://wiki.openstack.org/wiki/Freezer>

```

1  cat test_job.json
2
3  {
4      "job_actions": [
5          {
6              "freezer_action": {
7                  "action": "backup",
8                  "mode": "fs",
9                  "backup_name": "backup1",
10                 "path_to_backup": "/home/me/datadir",
11                 "container": "schedule_backups",
12                 "log_file": "/home/me/.freezer/freezer.log"
13             },
14             "max_retries": 3,
15             "max_retries_interval": 60
16         },
17         "job_schedule": {
18             "schedule_interval": "4 hours",
19             "schedule_start_date": "2015-08-16T17:58:00"
20         },
21     },
22     "description": "schedule_backups 6"
23 }
```



© Copyright 2017 Hewlett Packard Enterprise Development LP

13

## Example usage

To start the backup or restore job, you have to create the JSON backup description file. A sample backup job configuration file is shown on the right-hand side of the slide. Once the file is created, use the following command sequence to upload, list, display, and finally commit the backup job to the scheduler:

- Upload the job into the API:
 

```
freezer-scheduler -c node12 job-create
--file test_job.json
```
- List the newly created job:
 

```
freezer-scheduler -c node12 job-list
```
- Display the content of the job:
 

```
freezer-scheduler -c node12 job-get -j <job_id>
```
- Start the scheduler on the target node (commit the job):
 

```
freezer-scheduler -c node12 -i 15 -f ~/job_dir start
```



## Fuel (OpenStack® Environment Provisioning)

## What is the OpenStack® Fuel project?

- A tool that enables you to:
  - Provision multiple OpenStack® environments
  - Manage environments after deployment
- The key features are:
  - Hardware discovery
  - Hardware configuration in UI (networks and disk partitioning)
  - Support for non-HA and HA OpenStack® deployment configurations
  - Pre-deployment checks and network validation
  - Post-deployment checks and running a set of tests for validating deployed OpenStack®
  - Logs can be viewed in real-time through UI
  - Support for multiple OpenStack® distributions
- More information available at: <https://wiki.openstack.org/wiki/Fuel>

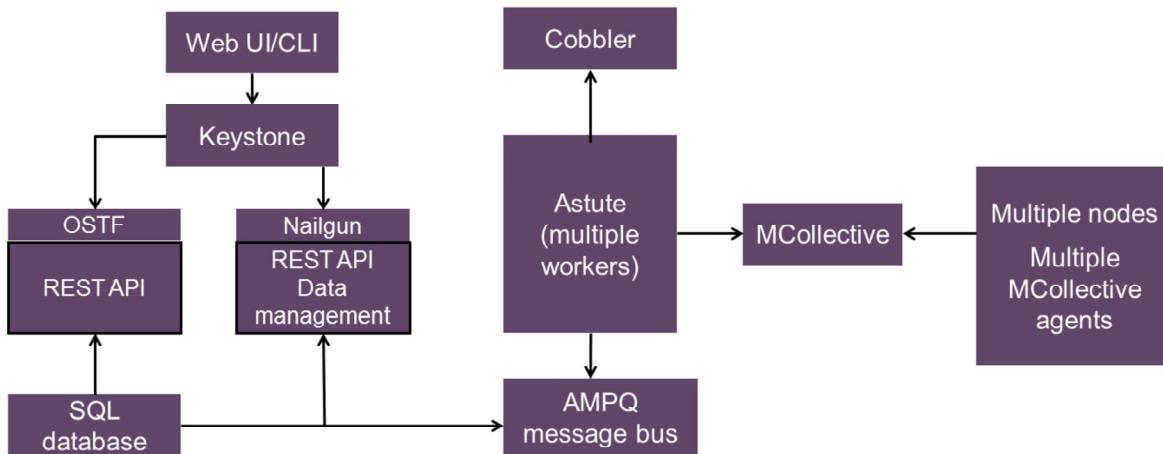


## What is the OpenStack® Fuel project?

Fuel is an open-source tool that enables you to easily and quickly provision multiple OpenStack® environments, as well as manage your environments after deployment. You can install Fuel and deploy your OpenStack® environment on a virtual platform such as Oracle VirtualBox or VMware vSphere for testing purposes, as well as on bare-metal hardware for production.

Fuel brings consumer-grade simplicity to streamline and accelerate the otherwise time-consuming, often complex, and error-prone process of deploying, testing, and maintaining various configuration flavors of OpenStack® at scale. Unlike other platform-specific deployment or management utilities, Fuel is an upstream OpenStack® project that focuses on automating the deployment and testing of OpenStack® and a range of third-party options, so it's not compromised by hard bundling or vendor lock-in.

## Fuel architecture



 Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

16

## Fuel architecture

Fuel is not monolithic. It consists of several independent components. Some of those components are Fuel-specific components, while others are third-party services like Cobbler, Puppet, MCollective, and so on. Some components can be reused separately from Fuel without any modifications and some require little tweaks.

**UI** is a single-page application, written in JavaScript. It uses bootstrap and backbone frameworks underneath.

**Nailgun** is the heart of the Fuel project. Nailgun is written in Python programming language. It implements REST API as well as deployment data management. It manages disk volumes configuration data, network configuration data, and any other environment-specific data that is necessary for successful deployment. It requires orchestration logic to build instructions for provisioning and deployment in a right order. Nailgun uses SQL database to store its data and AMQP service to interact with workers. The Fuel CLI provides even more functionality than UI.

**Astute** is another important component, which represents Nailgun's workers, and its function is to run certain actions according to the instructions provided from Nailgun. Astute is a layer which encapsulates all the details about interaction with a variety of services such as Cobbler, Puppet, shell scripts, and others, and provides universal asynchronous interface to those services. Depending on what you need to do, you can either manage a service directly via its native protocol (for example XML-RPC protocol is used for Cobbler) or you can use MCollective agents to perform specific tasks such as launching `puppet apply` on a remote node or running a script. Astute exchanges data with Nailgun via AMQP.

**Cobbler** is used as a provisioning service at the moment. There is a POC ready for move to Ironic, and production version is being implemented.

**Puppet** is the only deployment service at the moment. MCollective agent can be created to manage other configuration management frameworks, such as Chef, SaltStack, and so on.

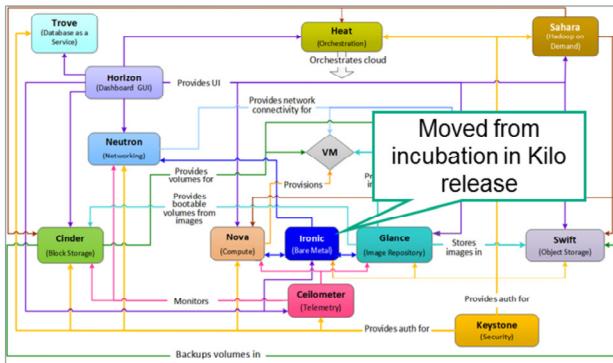
**MCollective agents** allow you to perform specific tasks like hard drives clearing, network connectivity probing, and similar.

**OSTF** (OpenStack Testing Framework, or Health Check) is a separate component which can be easily removed and reused without Fuel. It implements post-deployment verification of OpenStack. Its main goal is to verify a maximum of functionality taking minimum of time.



## Ironic (OpenStack® Bare Metal Provisioning)

## Ironic overview



- Is an OpenStack® service used to provision bare metal machines instead of virtual machines
- Is forked from the Nova baremetal driver
- Consists of metal hypervisor API and a set of plug-ins which interact with the baremetal hypervisors
- Uses PXE and IPMI in concert by default to provision and turn machines on or off
- Supports vendor-specific plug-ins which can implement additional functionality

**NOTE:** For a general overview, see <http://docs.openstack.org/developer/ironic/deploy/user-guide.html>.

## Ironic overview

Ironic is an OpenStack® project that provisions physical hardware rather than virtual machines (VMs). Ironic provides several reference drivers that leverage common technologies like the Preboot Execution Environment (PXE) and Intelligent Platform Management Interface (IPMI) to cover a wide range of hardware. The pluggable driver architecture of Ironic also enables you to add vendor-specific drivers for improved performance or for functionality that is not provided by the reference drivers.

If you think of traditional hypervisor functionality, such as creating a VM, enumerating virtual devices, managing the power state, loading an operating system (OS) on a VM, and so on, then you can think of Ironic as a hypervisor API gluing together multiple drivers, each of which implements some portion of that functionality with respect to physical hardware.

The OpenStack® Ironic project makes physical servers as easy to provision as VMs in cloud, which in turn opens up new avenues for enterprises and service providers.

The Ironic driver replaces the Nova “baremetal” driver that was added in the Grizzly through Juno releases. Ironic is available for use and is supported by the Ironic developers starting with the Juno release.

## Use cases for OpenStack® Bare Metal Provisioning

- Possible use cases for Ironic include:
  - High-performance computing clusters
  - Computing tasks that require access to hardware devices that cannot be virtualized
  - Database hosting (because some databases run poorly in a hypervisor)
  - Single tenant, dedicated hardware for performance, security, dependability, and other regulatory requirements
  - Rapidly deploying a cloud infrastructure



© Copyright 2017 Hewlett Packard Enterprise Development LP

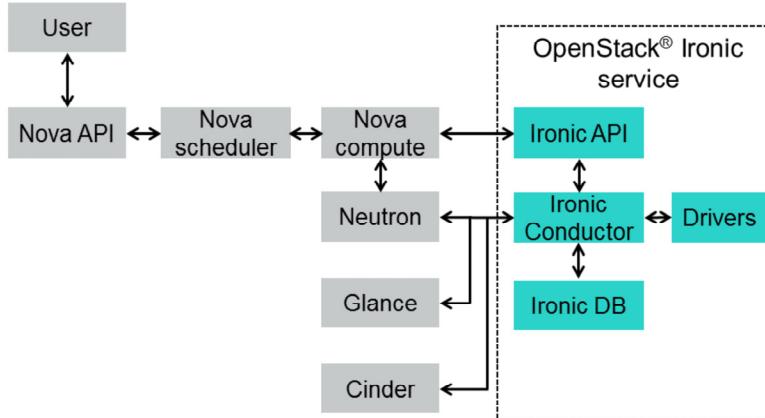
19

## Use cases for OpenStack® Bare Metal Provisioning

Here are a few use cases for baremetal (physical server) provisioning in cloud; there are doubtless many more:

- High-performance computing clusters
- Computing tasks that require access to hardware devices that cannot be virtualized
- Database hosting (because some databases run poorly in a hypervisor environment)
- Single-tenant, dedicated hardware for performance, security, dependability, and other regulatory requirements
- Rapidly deploying a cloud infrastructure

## The OpenStack® Ironic logical architecture



**NOTE:** The graphic above is an adapted version of the original diagram available at <http://docs.openstack.org/developer/ironic/deploy/user-guide.html>.

- The Ironic service is composed of the following components:
  - RESTful API service—The interface through which operators and other services interact with the managed baremetal servers
  - Conductor service—Does the majority of work and communicates with API services via RPC
  - Drivers that support heterogeneous hardware
  - Message Queue
  - Database for storing information about the resources (e.g. state of the conductors, nodes (physical servers), and drivers

## The OpenStack® Ironic logical architecture

The Ironic service is composed of these components:

- A RESTful API service, through which operators and other services can interact with the managed bare-metal servers
- A Conductor service that does the majority of the work
- Various drivers that support heterogeneous hardware
- A message queue
- A database for storing information about the resources

**Note:** The Conductor functionality is exposed by the API service. The Conductor and API services communicate by means of Remote Procedure Call (RPC).

With this logical architecture, a user request for booting an instance is passed to the Nova Compute service by means of the Nova API and Nova scheduler. The Compute service hands this request to the Ironic service, where the request passes from the Ironic API to the Conductor and then to a driver to successfully provision a physical server for the user.

Just as the Nova Compute service communicates with various OpenStack® services like Glance, Neutron, and Swift to provision a VM instance, the Ironic service communicates with the same OpenStack® services for image, networking, and other resource needs to provision a bare-metal instance.

## Key technologies for baremetal hosting

Technology	Description
PXE	PXE enables the system BIOS and NIC to bootstrap a computer from the network in place of a disk
DHCP	Using PXE, the BIOS uses DHCP to obtain an IP address for the network interface and to locate the server that stores the NBP
NBP	The NBP is responsible for loading the OS kernel in the memory so that the OS can be bootstrapped over a network
TFTP	In a PXE environment, TFTP is used to download NBP over the network, using information from the DHCP server
IPMI	IPMI is a method for managing systems that might be unresponsive or powered off; it uses only a network connection to the hardware rather than to an OS



© Copyright 2017 Hewlett Packard Enterprise Development LP

21

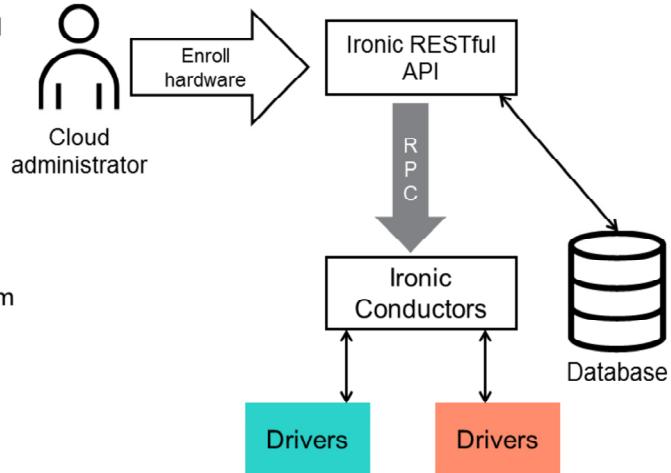
## Key technologies for baremetal hosting

Several technologies are used to enable baremetal hosting:

- **PXE:** The Preboot Execution Environment (PXE) is part of the Wired for Management (WfM) specification developed by Intel and Microsoft. PXE enables the system BIOS and network interface card (NIC) to bootstrap a computer from the network in place of a disk. Bootstrapping is the process by which a system loads an OS to local memory so it can be executed by the processor. This capability of allowing a system to boot over a network simplifies server deployment and server management for administrators.
- **DHCP:** The Dynamic Host Configuration Protocol (DHCP) 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. Using PXE, the BIOS uses DHCP to obtain an IP address for the network interface and to locate the server that stores the Network Bootstrap Program (NBP).
- **NBP:** The NBP is equivalent to the Grand Unified Bootloader (GRUB) and Linux Loader (LILO) loaders, which are traditionally used in local booting. Like the boot program in a hard-drive environment, the NBP is responsible for loading the OS kernel in the memory so that the OS can be bootstrapped over a network.
- **TFTP:** The Trivial File Transfer Protocol (TFTP) is a simple file transfer protocol that is generally used for the automated transfer of configuration or boot files between machines in a local environment. In a PXE environment, TFTP is used to download NBP over the network, using information from the DHCP server.
- **IPMI:** Intelligent Platform Management Interface (IPMI) is a standardized computer system interface used by system administrators for out-of-band management of computer systems and monitoring of their operation. It is a method for managing systems that might be unresponsive or powered off, using only a network connection to the hardware rather than to an OS.

## Ironic deployment architecture

- The Ironic RESTful API service is used to enroll the hardware that Ironic manages
- The Ironic Conductor does most of the work
- Multiple versions of the Conductor exist to support various classes of drivers and to manage failovers
- The API exposes a list of supported drivers and the names of the conductor hosts servicing them



## Ironic deployment architecture

The Ironic RESTful API service is used to enroll the hardware that Ironic manages. A cloud administrator usually registers the hardware, specifying attributes such as MAC addresses and IPMI credentials. The Ironic Conductor service does the majority of the work.

There can be multiple instances of the API service. Multiple instances of the Conductor service can be used to support various classes of drivers and to manage failovers. These instances of the Conductor service should be on separate nodes. Each conductor can run many drivers to operate heterogeneous hardware. The API exposes a list of supported drivers and the names of conductor hosts servicing them.

For security reasons, it is advisable to place the Conductor service on an isolated host, because it is the only service that requires access to both the data plane and the IPMI control plane.

## Prerequisites for baremetal deployment

- These prerequisites must be met before the deployment process begins:
  - Dependent packages must be configured on the compute node, such as `tftp-server`, `ipmi`, `syslinux`, and so on
  - Flavors must be created for the available hardware
    - Nova must know the flavor to boot from
  - Images must be made available in Glance
    - Some of the image types required for successful baremetal deployment are:
      - `bm-deploy-kernel`
      - `bm-deploy-ramdisk`
      - `user-image`
      - `user-image-vmlinuz`
      - `user-image-initrd`
    - Hardware must be enrolled via the Ironic RESTful API service



© Copyright 2017 Hewlett Packard Enterprise Development LP

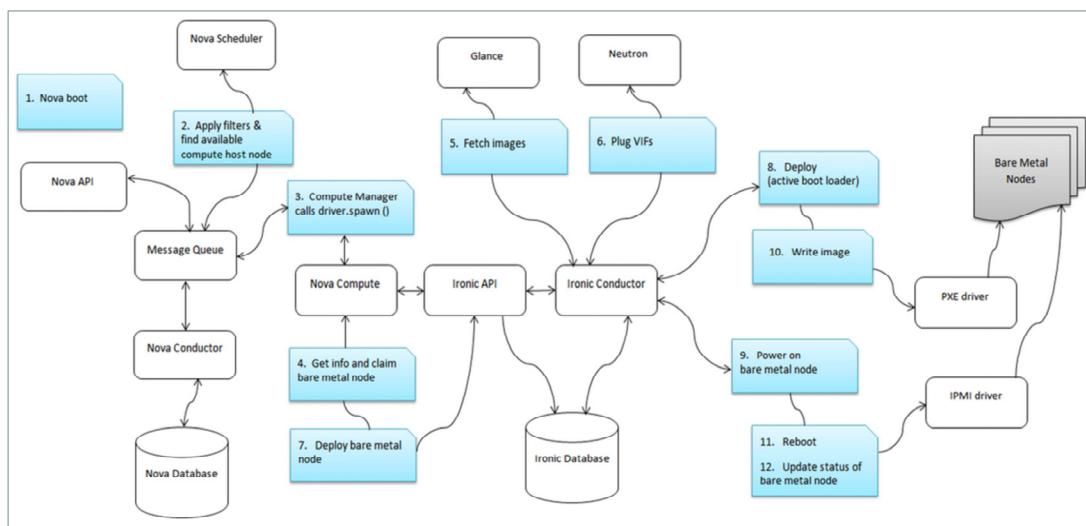
23

## Prerequisites for baremetal deployment

These prerequisites must be met before the deployment process begins:

- Dependent packages such as `tftp-server`, `ipmi`, `syslinux`, and so on, must be configured on the compute node for baremetal provisioning.
- Flavors must be created for the available hardware because Nova must know the flavor to boot from.
- Images must be made available in Glance; some of the image types required for successful baremetal deployment are:
  - `bm-deploy-kernel`
  - `bm-deploy-ramdisk`
  - `user-image`
  - `user-image-vmlinuz`
  - `user-image-initrd`
- Hardware must be enrolled by means of the Ironic RESTful API service.

## Baremetal deployment steps



NOTE: The graphic is obtained from <http://docs.openstack.org/developer/ironic/deploy/user-guide.html>.

Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

24

### Baremetal deployment steps

1. A boot instance request comes in through the Nova API and through the message queue to the Nova scheduler.
  2. The Nova scheduler applies the filter and finds the eligible compute node. The Nova scheduler uses flavor extra\_specs detail such as `cpu_arch`, `baremetal:deploy_kernel_id`, and `baremetal:deploy_ramdisk_id` to match the target physical node.
  3. The driver places a spawn task that contains all of the needed information, such as which image to boot from. It invokes the `driver.spawn` from the `virt` layer of Nova Compute.
  4. Information about the baremetal node is retrieved from the bare-metal database and the node is reserved.
  5. Images from Glance are pulled down to the local disk of the Ironic Conductor servicing the bare-metal node.
  6. Virtual interfaces are plugged in, and the Neutron API updates the DHCP port to support the PXE and TFTP options.
  7. The Nova Ironic driver issues a deploy request through the Ironic API to the Ironic Conductor that is servicing the bare-metal node.
  8. The PXE driver prepares TFTP bootloader.
  9. The IPMI driver issues a command to enable the network boot of a node and power it on.
  10. The DHCP boots the deploy ramdisk. The PXE driver copies the image over iSCSI to the physical node. It connects to the iSCSI end point, partitions volume, “dd” the image, and closes the iSCSI connection. At this point, the deployment is done. The Ironic Conductor then switches `pxe config` to the service mode and notifies the ramdisk agent about the successful deployment.
  11. The IPMI driver reboots the baremetal node.
- Note:** There are two power cycles during a baremetal deployment; the first is when the system is powered on and the images are deployed, as mentioned in step 9. The second time is after the images are deployed and the node is powered up.
12. The baremetal node status is updated and the node instance is made available.



## Magnum (Container Virtualization)

## What is Magnum?

- Magnum provides the API to manage application containers orchestration engines
- Keystone-compatible API
- Orchestration performed by the OpenStack® orchestration
- Leverages both Kubernetes and Docker as components
- More information available at: <https://wiki.openstack.org/wiki/Magnum>



© Copyright 2017 Hewlett Packard Enterprise Development LP

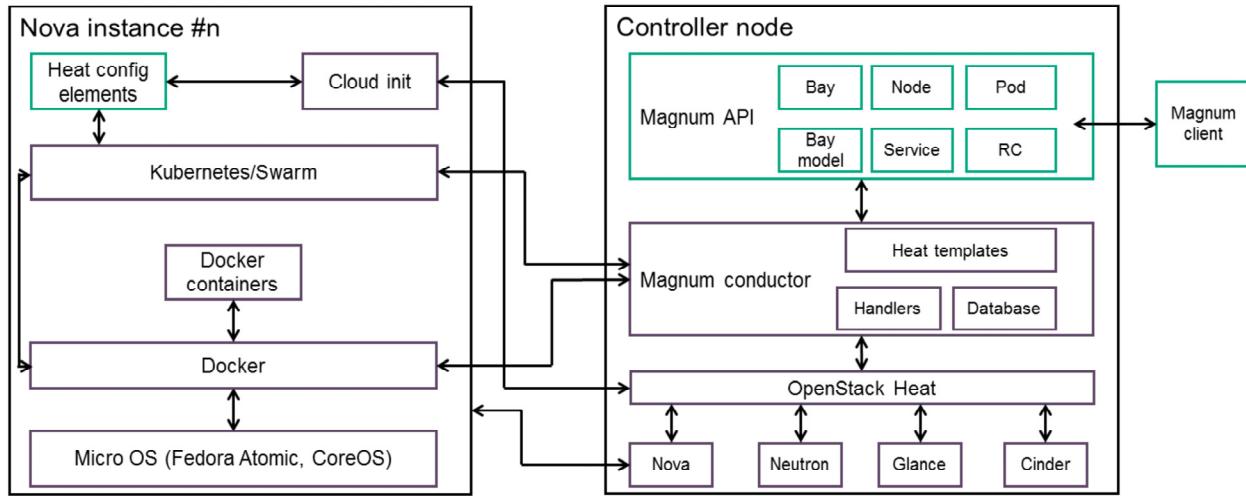
26

## What is Magnum?

Magnum is for OpenStack® cloud operators (public or private) who want to offer a self-service solution to provide a hosted container service to their cloud users. Magnum simplifies the required integration with OpenStack®. It allows cloud users who can already launch cloud resources (such as Nova instances, Cinder volumes, Trove databases, and so on) to create container clusters (bays) to run applications in an environment that provides advanced features that are beyond the scope of existing cloud resources.

The OpenStack® Magnum project provides a purpose-built API to manage the application container orchestration engines. Magnum offers an asynchronous API that is compatible with Keystone and a complete multi-tenancy implementation. It does not perform orchestration internally, and instead relies on OpenStack® Orchestration. Magnum leverages both Kubernetes and Docker as components.

## Magnum architecture



Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

27

## Magnum architecture

There are several different types of objects in the Magnum system:

- **Cluster:** A collection of node objects where work is scheduled
- **Cluster template:** An object stores template information about the cluster which is used to create new clusters consistently

Two binaries work together to compose the Magnum system. The first binary (accessed by the `python-magnumclient` code) is the `magnum-api` REST server. The REST server may run as one process or multiple processes. When a REST request is sent to the client API, the request is sent via AMQP to the `magnum-conductor` process. The REST server is horizontally scalable. At this time, the Conductor is limited to one process.

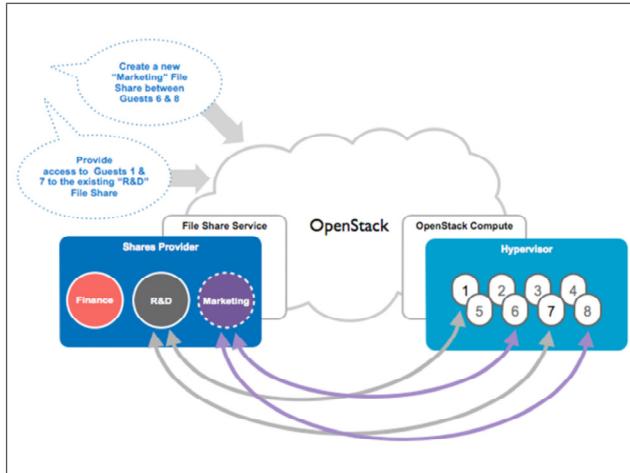


## Manila (Shared File System)

## What is Manila?

- OpenStack® service, originally based on Cinder
  - Cinder currently functions as the canonical storage provisioning service for block storage
  - The file share service provides coordinated access to shared or distributed file systems

**NOTE:** See <https://wiki.openstack.org/wiki/Manila> for links to the current project development status.



© Copyright 2017 Hewlett Packard Enterprise Development LP

29

## What is Manila?

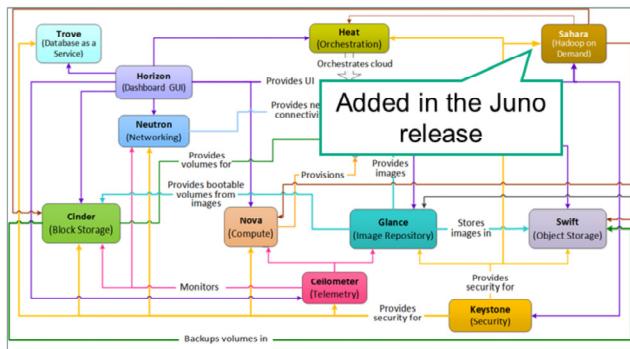
Cinder currently functions as the canonical storage provisioning control plane in OpenStack® for block storage, and it delivers a persistence model for instance storage. Similarly, the OpenStack® File Share service prototype provides coordinated access to shared or distributed file systems.

Although the primary consumption of file shares is across OpenStack® Compute instances, the Manila service is also accessible as an independent capability because it has been designed with the same modularity as other OpenStack® services. The design and prototype implementation provide extensibility for multiple backends to support vendor-specific or file system-specific nuances and capabilities, but it is sufficiently abstract to accommodate a variety of shared or distributed file system types.



## Sahara (Data Processing)

## Sahara functionality



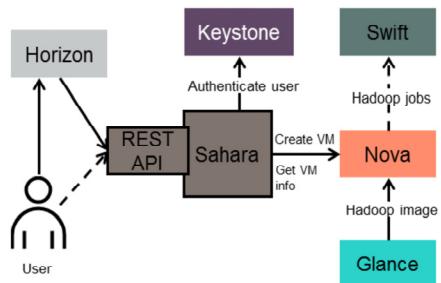
- Provides you with the ability to easily provision and manage Hadoop clusters on OpenStack®
- Managed through the REST API with a UI available as part of the OpenStack® Dashboard
- Support for different Hadoop distributions:
  - Pluggable system of Hadoop installation engines
  - Integration with vendor-specific management tools, such as Apache Ambari or Cloudera Management Console
- Provides predefined templates of Hadoop configurations with the ability to modify parameters

## Sahara functionality

Sahara provides you with the ability to easily provision and manage Hadoop clusters on OpenStack. It is managed through the REST API with a UI available as part of the OpenStack Dashboard.

Sahara supports a pluggable system of Hadoop installation engines and integrates with vendor specific management tools. It also provides predefined templates of Hadoop configurations and the ability to modify parameters.

## Sahara interactions with other OpenStack® projects



**NOTE:** The graphic above is an adapted version of the original diagram available at <http://docs.openstack.org/developer/sahara/overview.html>

- The Sahara project communicates with the following OpenStack® components:
  - **Horizon** provides a GUI with the ability to use all Sahara features
  - **Keystone** authenticates users and provides a security token that is used to work with OpenStack®
  - **Nova** is used to provision VMs for a Hadoop cluster
  - **Heat** orchestrates the required services for a Hadoop cluster
  - **Glance** is where Hadoop VM images are stored; each image contains an installed OS and Hadoop
  - **Swift** can be used as a storage for data that will be processed by Hadoop jobs
  - **Cinder** can be used as a block storage
  - **Neutron** provides the networking service
  - **Ceilometer** is used to collect measures of cluster usage for metering and monitoring purposes

## Sahara interactions with other OpenStack® projects

Sahara communicates with the following OpenStack® components:

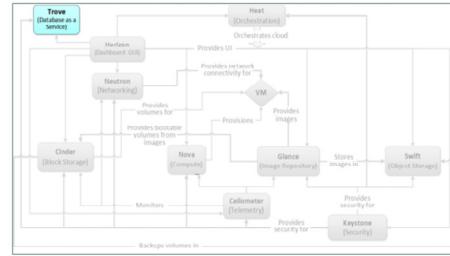
- **Horizon**—Provides a GUI with the ability to use all Sahara features
- **Keystone**—Authenticates users and provides a security token that is used to work with OpenStack®
- **Nova**—Used to provision VMs for a Hadoop cluster
- **Heat**—Orchestrates the required services for a Hadoop cluster
- **Glance**—This is where Hadoop VM images are stored; each image contains an installed OS and Hadoop
- **Swift**—Can be used as a storage for data that will be processed by Hadoop jobs
- **Cinder**—Can be used as a block storage
- **Neutron**—Provides the networking service
- **Ceilometer**—Used to collect measures of cluster usage for metering and monitoring purposes



## Trove (Database as a Service)

## Trove functionality

- Provides scalable and reliable cloud Database as a Service provisioning functionality for both relational and non-relational database engines
- Interacts with other OpenStack® components purely through the API
- The Trove RESTful API allows for:
  - Spinning up and resizing instances
  - Creating replicas
  - Adding users and databases, and managing grants
  - Managing database backups
  - Changing database configurations
- VM instance deployed on demand with a fully functional database
- Cloud users and database administrators can provision and manage multiple database instances as needed



Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

34

## Trove functionality

Database-as-a-Service (DBaaS) is generally of one of two different forms:

- A virtual machine image of a database installation, where you still need to configure the database for your needs.
- A vendor-supplied database service, which does not directly expose you to the physical or virtual instances behind the service itself. The vendor manages the service including security, elasticity, and multi-tenancy.

Each database server under management holds a portion of the complete data set, referred to as "a slice." Each table in the application database is distributed (or sliced) among different database servers depending on its primary purpose in the setup. Tables containing supporting information (sometimes referred to as "dimension tables") can "broadcast" across all MySQL servers to facilitate easy joining with other tables. Optionally, dimension tables may be sliced and distinct subsets can be stored on each database server.

Trove is the OpenStack® Database as a Service. It is designed to run entirely on OpenStack®, with the goal of allowing you to quickly and easily utilize the features of a relational database without the burden of handling complex administrative tasks. Cloud users and database administrators can provision and manage multiple database instances as needed. Initially, the service focuses on providing resource isolation at high performance while automating complex administrative tasks including deployment, configuration, patching, backups, restores, and monitoring.

Trove is designed to support a single-tenant database within a Nova instance. There are no restrictions on how Nova is configured, since Trove interacts with other OpenStack® components through the API.

Trove's mission is to provide a scalable and reliable cloud Database as a Service provisioning functionality for both relational and non-relational database engines, and to continue improving its fully-featured and extensible open source framework.

## Trove use cases

- Database users
  - From the beginning, the Trove service has offered a highly available way to deploy database servers
  - Creating a new database does not involve a system administrator—the whole process is automated and controlled by end users
- Database administrators
  - Administration is made easy by consolidating database instances on dedicated VMs
  - Administration is simplified through isolation database services, eliminating the side effects of running unrelated services on database servers
  - Database backup and configuration are controlled and can be performed in parallel across the infrastructure by a single command
- Mission-critical environments
  - Benefit from database replication technologies (for example, MySQL replication)
  - Benefit from high-availability OpenStack® services



© Copyright 2017 Hewlett Packard Enterprise Development LP

35

### Trove use cases

Trove currently enables you to provision a database within a single instance, manage users and databases, back up and restore data, resize instances, and check quota usage. Running entirely on OpenStack®, Trove can be used in a variety of scenarios and configurations.

#### Database users

From the beginning, the Trove service has offered a highly available way to deploy database servers. Creating a new database does not involve a system administrator—the whole process is automated and controlled by end users.

#### Database administrators

Administration has been made easy by consolidating database instances on dedicated virtual machines (VMs). Administration is simplified through isolation database services, which eliminated the side effects of running unrelated services on database servers.

Database backup and configuration are controlled and can be performed in parallel across the infrastructure through a single command.

#### Mission-critical environments

Mission-critical environments directly benefit from database replication technologies such as MySQL replication. Based completely on the OpenStack® platform, the Trove service leverages high-availability features, making disaster recovery and business continuity possible and easy to implement.

## Trove terminology

Term	Definition
Datastore	<ul style="list-style-type: none"> <li>– An abstraction of the underlying database</li> <li>– Current implementations are for MySQL, MongoDB, PostgreSQL, Memcached, and more</li> </ul>
Datastore version	<ul style="list-style-type: none"> <li>– Represents a released version of a datastore (for example, 5.5 for MySQL)</li> <li>– Provides linkage to a guest image stored in Glance</li> </ul>
Instance	<ul style="list-style-type: none"> <li>– One incarnation of a datastore version</li> <li>– Managed by Nova and using persistent volumes from Cinder</li> <li>– Runs a full OS plus trove-guestagent</li> </ul>
Configuration group	<ul style="list-style-type: none"> <li>– Represents a collection of datastore-specific configuration parameters</li> <li>– Used to customize the configuration of a single instance or a group of instances</li> </ul>
Database	<ul style="list-style-type: none"> <li>– The database running on the datastore</li> <li>– Several databases can run in one instance</li> <li>– Managed through the OpenStack® APIs and trove-guestagent</li> </ul>



© Copyright 2017 Hewlett Packard Enterprise Development LP

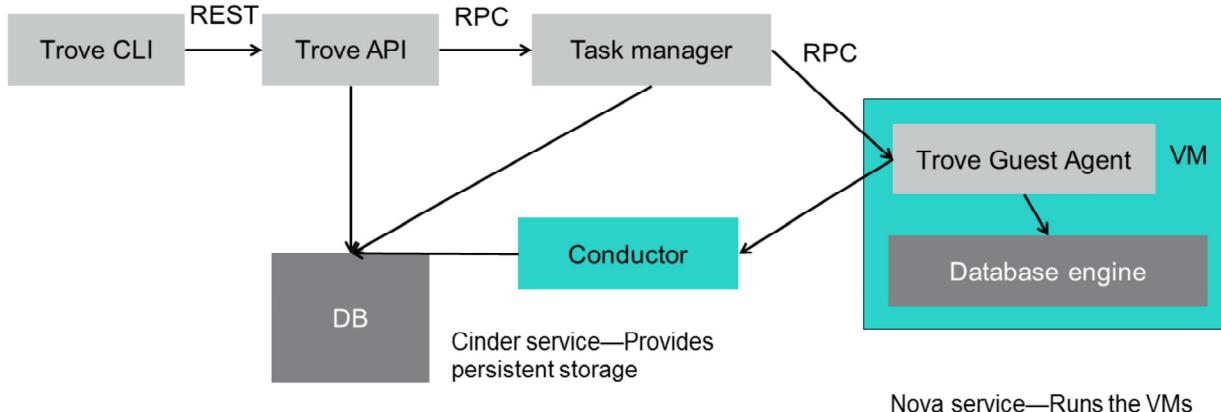
36

## Trove terminology

To fully understand the function of the Trove service, you should be familiar with the following terminology:

- **Datastore**: An abstraction of the underlying database. The current implementations are for MySQL, MongoDB, PostgreSQL, Memcached, and others.
- **Datastore version**: Represents a released version of a datastore (such as 5.5 for MySQL), and provides linkage to a guest image stored in Glance.
- **Instance**: One incarnation of a datastore version. Instances are managed by Nova and use persistent volumes from Cinder. They run a full operating system (OS) plus trove-guestagent.
- **Configuration group**: Represents a collection of datastore-specific configuration parameters. They are used to customize the configuration of a single instance or a group of instances.
- **Database**: The database running on the datastore. Several databases can run in one instance. They are managed through the OpenStack® APIs and trove-guestagent.

## Trove architecture overview



Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

37

### Trove architecture overview

Trove offers a Representational State Transfer (REST) API to provision instances and manage the service itself, leveraging the API to plug into OpenStack®. The Trove service is built on the following OpenStack® services:

- **Keystone**: Used to provide authorization, authentication, and access control (AAA)
- **Neutron**: Used to manage networking
- **Cinder**: Providing persistent storage service
- **Nova**: Compute service to support virtual machines

The Trove Conductor subservice oversees all of the individual guest DBaaS instances, managing the message bus communication and routing commands, maintaining the overall state of the distributed database service, and coordinating critical functions such as backups and restores.

## Trove components

Component	Description
trove-api	A service that provides a RESTful API that supports JSON and XML to provision and manage Trove instances
trove-taskmanager	A service that provisions and manages the lifecycle of OS instances and that performs operations on the database instance
trove-guestagent	A service that runs within the guest instance, and manages and performs operations on the database itself
trove-conductor	A service that runs on the host and receives messages from guest instances to update information on the host



© Copyright 2017 Hewlett Packard Enterprise Development LP

38

## Trove components

Trove relies on four primary components to provide DBaaS:

- **trove-api:** The trove-api service provides a RESTful API that supports JavaScript Object Notation (JSON) and XML to provision and manage Trove instances.
- **trove-taskmanager:** The trove-taskmanager service does the complex work regarding provisioning instances, managing the lifecycle of instances, and performing operations on the database instance.
- **trove-guestagent:** The trove-guestagent is a service that runs within the guest instance. It manages and performs operations on the database itself. The guest agent listens for Remote Procedure Call (RPC) messages through the message bus and performs the requested operation.
- **trove-conductor:** The Conductor service runs on the host. It receives messages from guest instances to update information on the host, such as instance statuses and the current status of a backup. With the Conductor service, guest instances do not need a direct connection to the host database. The Conductor service listens for RPC messages through the message bus and performs the relevant operation.

## Trove-supported databases

- **SQL-based:**

- MySQL
- CouchDB
- PostgreSQL

- **NoSQL:**

- MongoDB
- Casandra
- Redis

- More information available at: <https://wiki.openstack.org/wiki/Trove>



© Copyright 2017 Hewlett Packard Enterprise Development LP

39

## Trove-supported databases

The list of databases supported by Trove is given on the slide above.



## Zaqar (Multi-Tenant Cloud Messaging)

## What is Zaqar?

A multi-project cloud messaging service for web developers

- Features a fully RESTful API
- Has an efficient messaging engine designed for scalability and security
- Has a firewall-friendly, HTTP-based API with Keystone support
- Leverages multi-tenant queues that are based on Keystone project IDs
- Supports several common patterns, including event broadcasting, task distribution, and point-to-point messaging
- Has a component-based architecture that supports custom backends and message filters
- Provides an efficient reference implementation for low latency and high throughput (dependent on the backend)
- Is highly available and horizontally scalable

**NOTE:** See <https://wiki.openstack.org/wiki/Zaqar> for links to the current project development status.



© Copyright 2017 Hewlett Packard Enterprise Development LP

41

## What is Zaqar?

Zaqar is a multi-tenant cloud messaging service for web developers. It combines the ideas pioneered by the Amazon SQS product with additional semantics to support event broadcasting.

The service features a fully RESTful API that developers can use to send messages between various components of their Software as a Service (SaaS) and mobile applications, by using a variety of communication patterns. Underlying this API is an efficient messaging engine that was designed for scalability and security.

Other OpenStack® components can integrate with Zaqar to bring events to users and to communicate with guest agents that run in the “over-cloud” layer. Cloud operators can leverage Zaqar to provide the equivalents of the Amazon Simple Queue Service (SQS) and Simple Notification Service (SNS) to their customers.

## Zaqar design principles

- Zaqar was designed with the following guidelines:
  - **Component-based architecture:** Quickly add new behaviors
  - **High availability and scalability:** Scale to large workloads
  - **Fault tolerance:** Avoid cascading failures with isolated processes
  - **Recoverability:** Easily diagnose, debug, and rectify failures
  - **Open standards:** Be a reference implementation for a community-driven design



© Copyright 2017 Hewlett Packard Enterprise Development LP

42

## Zaqar design principles

Zaqar, like all OpenStack® projects, was designed with the following guidelines in mind:

- **Component-based architecture:** Quickly add new behaviors
- **High availability and scalability:** Scale to large workloads
- **Fault tolerance:** Avoid cascading failures with isolated processes
- **Recoverability:** Easily diagnose, debug, and rectify failures
- **Open standards:** Be a reference implementation for a community-driven design



**Hewlett Packard  
Enterprise**



Hewlett Packard  
Enterprise

# Fundamentals of OpenStack® Technology

Module 13—OpenStack® Deployment Planning

H6C68S E.01

© Copyright 2017 Hewlett Packard Enterprise Development LP



## Learning objectives

After completing this module, you should be able to:

- Discuss OpenStack® deployment considerations
- Identify the cloud services and offerings provided by HPE



© Copyright 2017 Hewlett Packard Enterprise Development LP

2

## Learning objectives

After completing this module, you should be able to:

- Discuss OpenStack® deployment considerations
- Identify the cloud services and offerings provided by HPE



# OpenStack® deployment considerations

## Cloud controller

- A single node
- Hosts a number of services
  - Image management
  - Dashboard
  - Authentication and authorization
  - Databases
  - Message queue
  - API endpoints
- OpenStack® services can be deployed on separate nodes for scalability



© Copyright 2017 Hewlett Packard Enterprise Development LP

4

### Cloud controller

Each cloud is managed by a cloud controller. A cloud controller is a node that hosts the databases, message queue service, authentication and authorization service, image management service, user dashboard, and API endpoints.

The cloud controller provides centralized management for a multi-node OpenStack® deployment. Typically, the cloud controller manages authentication and sends messaging to all the OpenStack® services through a message queue.

The cloud controller has a collection of nova-\* components that:

- Represent the global state of the cloud
- Communicate with the various OpenStack® services such as authentication
- Maintain information about the cloud in a database
- Communicate to all compute nodes and storage workers through a queue
- Provide API access

Each service running on a designated cloud controller can be broken out into separate nodes for scalability or availability.

## Cloud controller hardware considerations

- Many considerations when scaling your cloud controller:
  - The number of **instances** you expect to support
    - Ensure that the database server scales accordingly
  - The number of **compute nodes** in the cloud
    - Impacts the message queuing performance
  - The number of **users** to access compute and storage services
    - Impacts the CPU load
  - API services only or dashboard access as well?
    - The dashboard makes many more requests than mere API access
  - Authentication services
    - External authentication, such as LDAP or Active Directory, and impacts to the CPU load
  - Be aware of core count requirements



© Copyright 2017 Hewlett Packard Enterprise Development LP

5

### Cloud controller hardware considerations

When specifying cloud controller hardware, a good starting point is to consider the same type of hardware as you would use for a compute node. However, you have to consider the load on various services—not only the CPU load, but the network and memory load as well.

You can use virtual machines (VMs) for all or some of the services that the cloud controller manages, such as the message queuing.

To size the cloud controller correctly and determine whether to virtualize any part of it, you should estimate:

- The number of instances that you expect to run
- The number of compute nodes that you have
- The number of users who will access the compute or storage services
- Whether users interact with your cloud through the Representational State Transfer (REST) API or the dashboard
- Whether users authenticate against an external system such as the Lightweight Directory Access Protocol (LDAP) or Active Directory
- How long you expect single instances to run

## Separating services

- Start with a single controller node and scale as required, separating services on different nodes
  - Glance services on Swift proxy servers
    - Put the image management on hardware as backend storage management
  - Central database server
    - Makes database server updates easier
    - Make it disaster- or fault-tolerant
  - Consider running each service on a VM
    - Easier to scale individual services
- If you are using multiple Nova or Swift proxy servers, consider using an external load balancer
- Cluster the message queuing services to avoid a single point of failure
- Load balance the Nova API service
- Use fast network connections such as 10Gb NICs and bonded NICs



© Copyright 2017 Hewlett Packard Enterprise Development LP

6

### Separating services

It is worth considering whether to run all of your cloud controller services on a single node or to spread them across several nodes. Start by estimating the expected load on the cloud controller and then scale out as required, based on your needs.

Glance can benefit from running on physical hardware and having good connectivity to the object storage backend.

Consider using a central dedicated server to provide the databases for all services. This model can simplify operations by isolating database server updates and allowing slave database servers for failover.

You can run various services on separate VMs. Then you can monitor and tune each VM to the service it hosts.

Consider placing services across several servers, using a load balancer to switch between them.



## HPE cloud offerings

## The HPE POV: Hybrid delivery with an HPE Cloud

### Cloud delivery model: Build, consume, or both



Traditional IT



Private cloud



Managed cloud



Professional services

- Advisory
- Strategy
- Application

- Transformation
- Design
- Implementation

- Operation
- Education

### Software-enabled service management

- Analytics
- Metrics
- Dashboard

- Billing
- Chargeback
- Asset management

HPE Cloud common architecture with the HPE Cloud OS

 Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

8

## The HPE POV: Hybrid delivery with an HPE Cloud

A hybrid cloud approach provides the means to address various IT service delivery challenges facing IT organizations today in the most efficient and cost effective way. This means taking advantage of the capabilities offered across the private cloud, managed cloud, and public cloud models and traditional IT, considering the various deployment models as a continuum of service level agreements (SLAs), and selecting an optimal mix of delivery models to achieve speed and agility.

Achieving a hybrid cloud models means looking at the portfolio of applications in your organization, segmenting them according to their respective SLA requirements, and then aligning the applications to the appropriate deployment model, thus creating the most optimal use of your internal and external resources.

In the hybrid world, the role of the chief information officer and IT organizations expands from the traditional builder of services to a build and broker of services, creating a seamless experience for users independent of the service source.

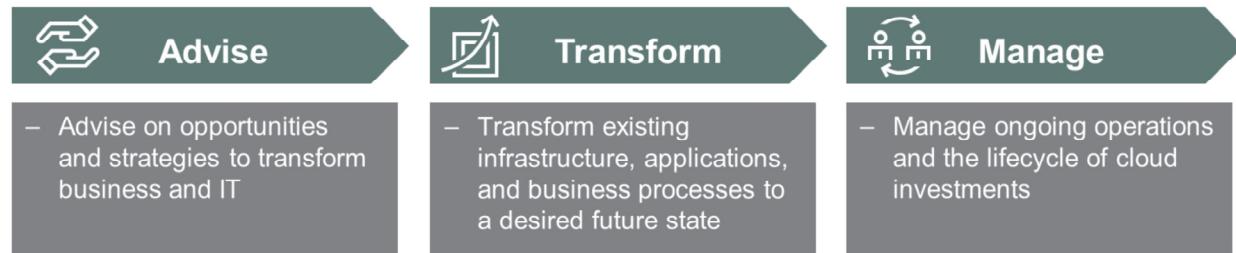
The hybrid cloud requires a broad portfolio of hardware, software, and services working together to drive outcomes.

A narrow focus like that of some competitors locks you into their ecosystem, and also the outcomes that matter to them – not to you.

## HPE Converged Cloud Professional Services

Cloud experts you can rely on

- HPE provides a comprehensive portfolio of cloud professional services to advise, transform, and manage your cloud journey
- HPE takes an end-to-end lifecycle approach to address your needs in the areas of people, processes, and technology



### HPE Converged Cloud Professional Services—Cloud experts you can rely on

Many organizations are embracing hybrid delivery models to speed innovation, increase agility, and lower operating costs. However, disparate architectures, underperforming SLAs, and fragmented support can slow the adoption of a hybrid model that includes private, public, and managed clouds.

HPE Technology Services provides you with a single contact for your hybrid cloud design, implementation, and support that is personalized for your unique needs, delivering a proactive and cohesive cloud solution.

## HPE Converged Cloud management offerings

A complete portfolio for cloud IT operations management

### Cloud management

#### Service Lifecycle Management

HPE Cloud Service Automation

#### Automation

- HPE Operations Orchestration
- HPE Server Automation
- DB and Middleware Automation

#### Security

- HPE TippingPoint
- HPE Fortify
- HPE ArcSight

#### Business Management

- HPE Asset Manager
- HPE Executive Scorecard

#### Performance Management

- HPE Business Service Management

#### Compliance

- HPE Server Automation

#### Service Management

- HPE Service Manager

#### Capacity and Resource Management

- HPE Matrix Operating Environment
- Adaptive Computing Moab

#### Application Service Models

- HPE Cloud Maps

### HPE Converged Cloud management offerings—A complete portfolio for cloud IT operations management

The HPE Converged Cloud offers a complete portfolio of solutions and services to meet your needs. Across private, managed, public, and hybrid clouds, HPE cloud solutions are built on a common foundation of HPE innovation and enterprise experience:

- **Choice**—Whether you are building onto existing infrastructure or from the ground up, the HPE multi-platform, multi-hypervisor cloud solutions are designed with open, standards-based technology with no vendor lock-in.
- **Confidence**—HPE provides a complete model with the right security and reliability levels across all delivery platforms. You can trust HPE to scale with you as your business evolves and transforms.
- **Consistency**—With the only common architecture across private, managed, and public clouds in the industry, rework is unnecessary. Your customers will appreciate the same application experience regardless of the backend.

## HPE Converged Cloud Professional Services portfolio



### Advise

#### Cloud advisory

- Deliver insight and knowledge and identify opportunities to begin a cloud journey

#### Cloud strategy

- Develop a business case, a transformation plan, and a multi-year road map to a future state across a hybrid environment



### Transform

#### Application transformation to Cloud

- Design, develop, migrate, and test applications and business processes to exploit cloud benefits

#### Cloud design

- Best practices and frameworks to build detailed architectures and designs for cloud solutions

#### Cloud implementation

- Build, integrate, migrate, and deploy cloud solutions



### Manage

#### Cloud operation

- Achieve best-in-class operational efficiency of your cloud environment, leveraging the unique knowledge of HPE experts and HPE global infrastructure

#### Cloud education

- Train and certify your IT staff and third-party partners to help them design, integrate, and administer cloud solutions

## HPE Converged Cloud Professional Services portfolio

HPE offers a Cloud Services portfolio that matches the journey of each client and their unique requirements across the entire journey.

To provide you with insights and advice, HPE offers Cloud Advisory Services. These services identify opportunities to embrace the cloud based on business needs. With Cloud Strategy Services, HPE helps you develop a business case, a strategy, and a transformation plan to get you to the desired future state.

Based on that transformation plan, HPE helps you modernize and transform your applications, business processes, and infrastructure to take advantage of the cloud. Using the work from Cloud Design Services, HPE helps you build and deploy the appropriate solutions through Cloud Implementation Services.

As cloud services are deployed and updated over time, HPE Cloud Operation Services help you achieve best-in-class operational efficiency and support the services lifecycle end to end. With HPE Cloud Education Services, HPE can help to train and certify IT staff and partners to ensure they have the latest knowledge and skill set to continue to maintain and manage their cloud investments.

## Module summary

- In this module:
  - Discuss OpenStack® deployment considerations
  - Identify the cloud services and offerings provided by HPE



© Copyright 2017 Hewlett Packard Enterprise Development LP

12

## Module summary

In this module:

- Discuss OpenStack® deployment considerations
- Identify the cloud services and offerings provided by HPE

# Learning check

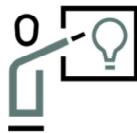


Hewlett Packard  
Enterprise

© Copyright 2017 Hewlett Packard Enterprise Development LP

13

## Learning check



How does the number of users impact the cloud controller?

- A. It impacts the message queueing performance
- B. It impacts the CPU load
- C. It impacts the amount of available disk space

## Learning check

How does the number of users impact the cloud controller?

- A. It impacts the message queueing performance
- B. It impacts the CPU load
- C. It impacts the amount of available disk space

## Learning check answer



How does the number of users impact the cloud controller?

- A. It impacts the message queueing performance
- B. It impacts the CPU load**
- C. It impacts the amount of available disk space



© Copyright 2017 Hewlett Packard Enterprise Development LP

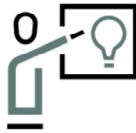
15

## Learning check answer

How does the number of users impact the cloud controller?

- A. It impacts the message queueing performance
- B. It impacts the CPU load**
- C. It impacts the amount of available disk space

## Learning check



When planning for service separation, why would you consider running each service on a VM?

- A. To make it easier to scale individual services
- B. To make services disaster- or fault-tolerant
- C. To make the database server updates easier



© Copyright 2017 Hewlett Packard Enterprise Development LP

16

## Learning check

When planning for service separation, why would you consider running each service on a VM?

- A. To make it easier to scale individual services
- B. To make services disaster- or fault-tolerant
- C. To make the database server updates easier

## Learning check answer



When planning for service separation, why would you consider running each service on a VM?

- A. To make it easier to scale individual services
- B. To make services disaster- or fault-tolerant
- C. To make the database server updates easier

## Learning check answer

When planning for service separation, why would you consider running each service on a VM?

- A. To make it easier to scale individual services
- B. To make services disaster- or fault-tolerant
- C. To make the database server updates easier



## Take control of your future with training from HPE Education Services

Visit: [www.hpe.com/ww/learn](http://www.hpe.com/ww/learn)

You can also follow us on Facebook and Twitter – use the links below and join the HPE Education Services Virtual Community



[www.facebook.com/HPEEnterpriseBusiness/app\\_11007063052](http://www.facebook.com/HPEEnterpriseBusiness/app_11007063052)

[www.facebook.com/HPTechnologyServices](http://www.facebook.com/HPTechnologyServices)

[www.twitter.com/HPEducation](http://www.twitter.com/HPEducation)



© Copyright 2017 Hewlett Packard Enterprise Development LP

18



**Hewlett Packard  
Enterprise**

© Copyright 2017 Hewlett Packard Enterprise Development LP 

