zstack Documentation

Release 0.6

zstack.org

March 15, 2016

Contents

1 Introduction 1

2 Chapters 3

CHAPTER 1

Introduction

ZStack is an open source software that manages compute nodes, networks, and storage to provide infrastructure as a service(IaaS) solution, written in Java and Python.

This documentation is a full reference of all ZStack features. If you haven't installed ZStack and tried out several tutorials, please visit our web site for installation and tutorials.

Chapters in this documentation are arranged in sections of:

- **Overview**: gives your a brief background of the topic.
- **Inventory**: explains the data model of the resource (e.g. zone, virtual machine), which usually starts with a table listing properties of the resource, and is followed by detailed explanations of properties that are not straightforward.
- **Operations**: explains every API manipulating the resource. APIs are explained in examples of ZStack command tool that you will see in chapter 3.
- **Global Configurations**: explains every global configuration that can be applied to the resource, if there is any.
- **System Tags**: explains every system tag that can be applied to the resource, if there is any.

We recommend users to start with the chapter *Introduction* and read at least chapters *Resource Model*, *Command Line Tool*, and *Query* all of which are important for your daily use of ZStack. For other chapters, you can use them as references when you need, for example, looking up chapter *Virtual Machine* when you want to find out the command for creating a VM.

CHAPTER 2

Chapters

2.1 Introduction

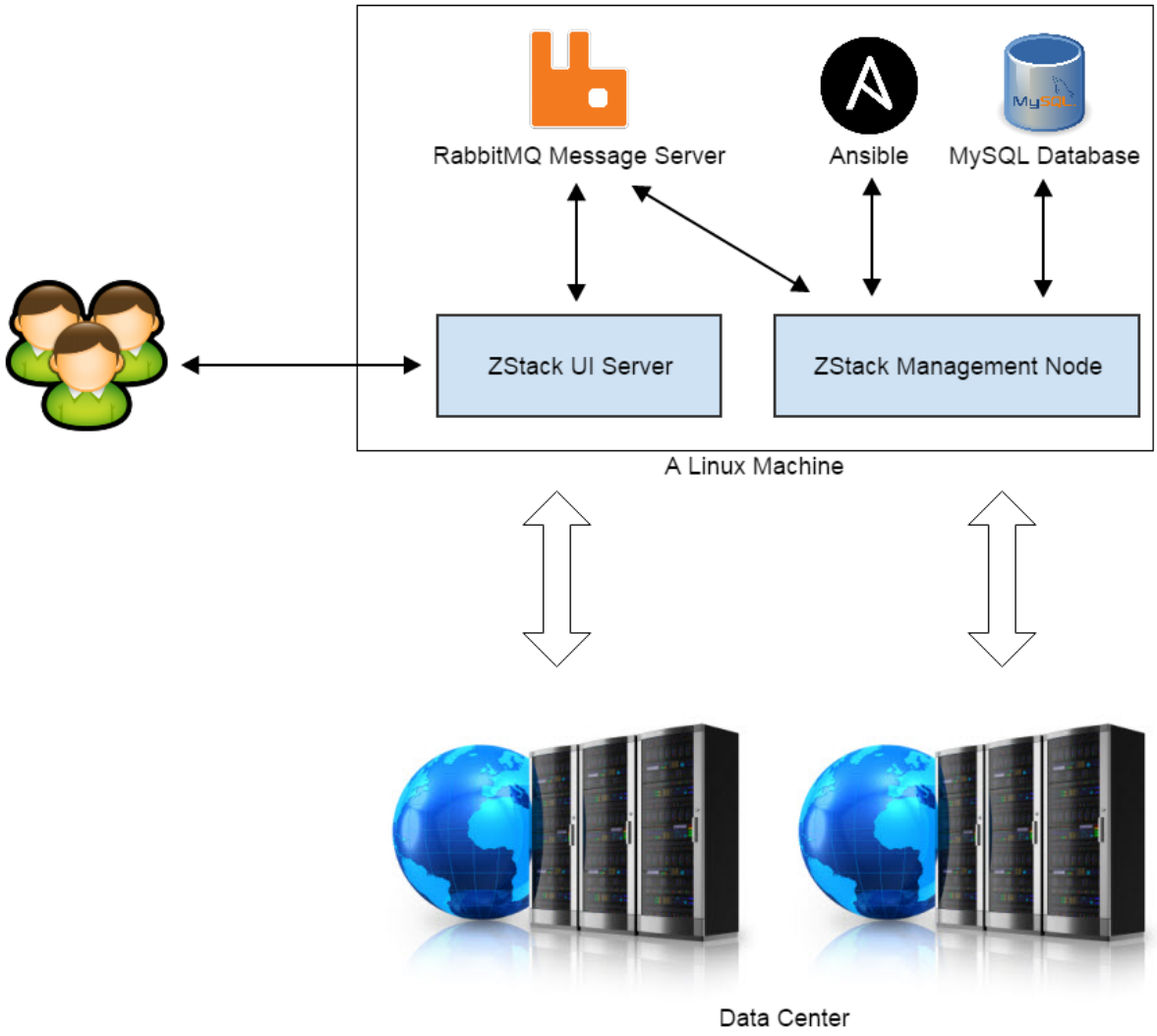
Table of contents

- *Introduction*
  - *Overview*
    - *The Setup of A Single Management Node*
    - *A Deployment of Multiple Management Nodes*
    - *ZStack's World View of A Cloud*

2.1.1 Overview

Depending on the scale of a cloud, a ZStack setup can be as simple as a single Linux machine running one ZStack management node, or a cluster of Linux servers running multiple ZStack management nodes.

The Setup of A Single Management Node



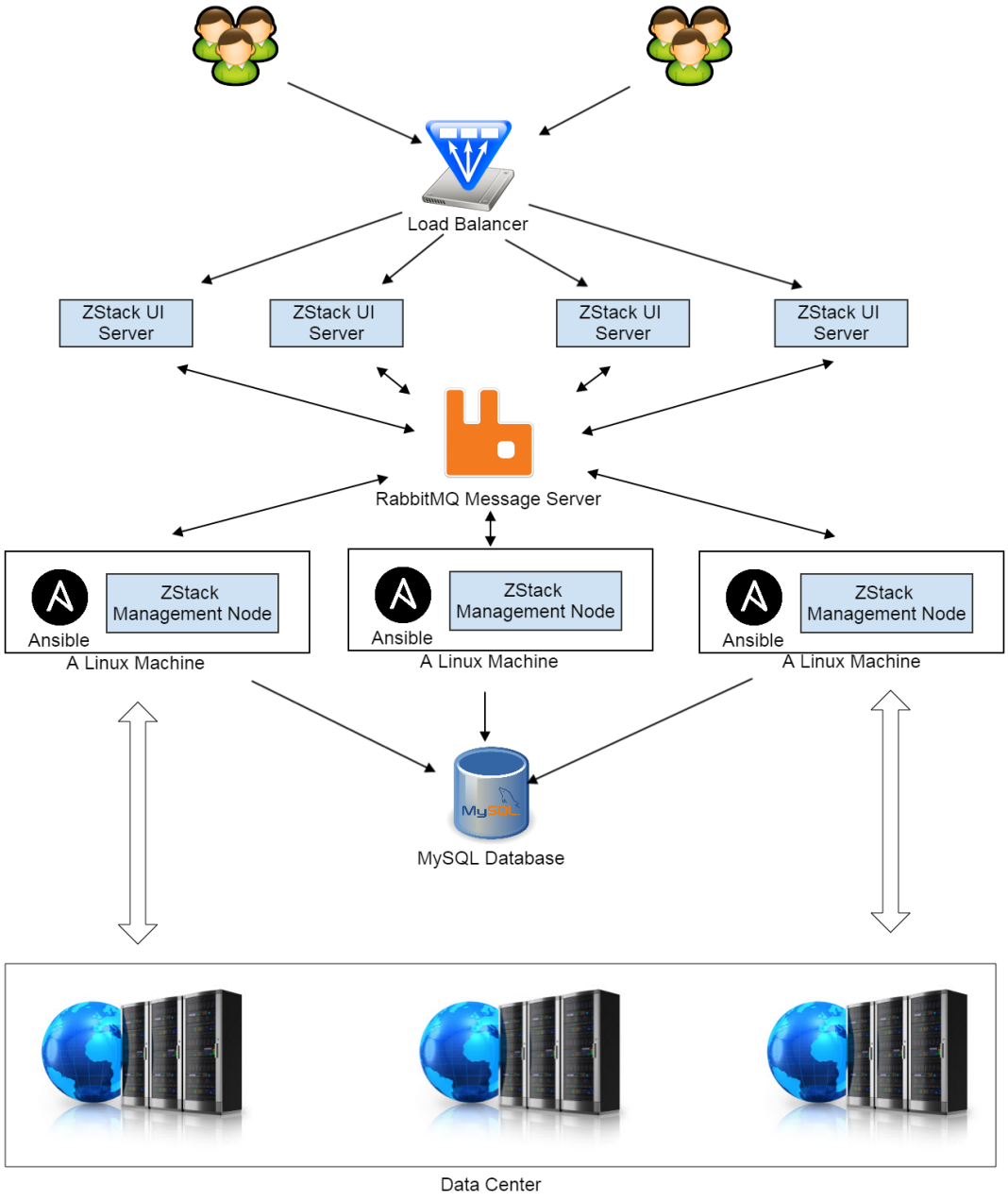
In the simplest setup, all ZStack software and third party dependencies are installed on a single Linux server. A typical setup includes five parts:

- RabbitMQ Message Server: The central message bus ZStack services use for communication.
- MySQL Database: The database ZStack stores metadata for resources in the cloud.
- Ansible: The configuration management tool ZStack uses to remotely deploy and upgrade agents.
- ZStack Management Node: The main process encompassing all ZStack orchestration services.
- ZStack UI Server: A web server providing user interface for end users.

Besides, several python agents, which need deploying to local or remote machines at runtime, are packaged in the WAR file of ZStack management node and are deployed using Ansible.

Because of ZStack's asynchronous architecture, a single management node is normally enough to manage a big cloud that may have tens of thousands of physical servers, hundreds of thousands of virtual machines(virtual machine is referred as VM in future chapters), and tens of thousands of concurrent API requests. However, in case of high availability and scaling out for a super large cloud, a setup of multiple management nodes is still valuable.

A Deployment of Multiple Management Nodes



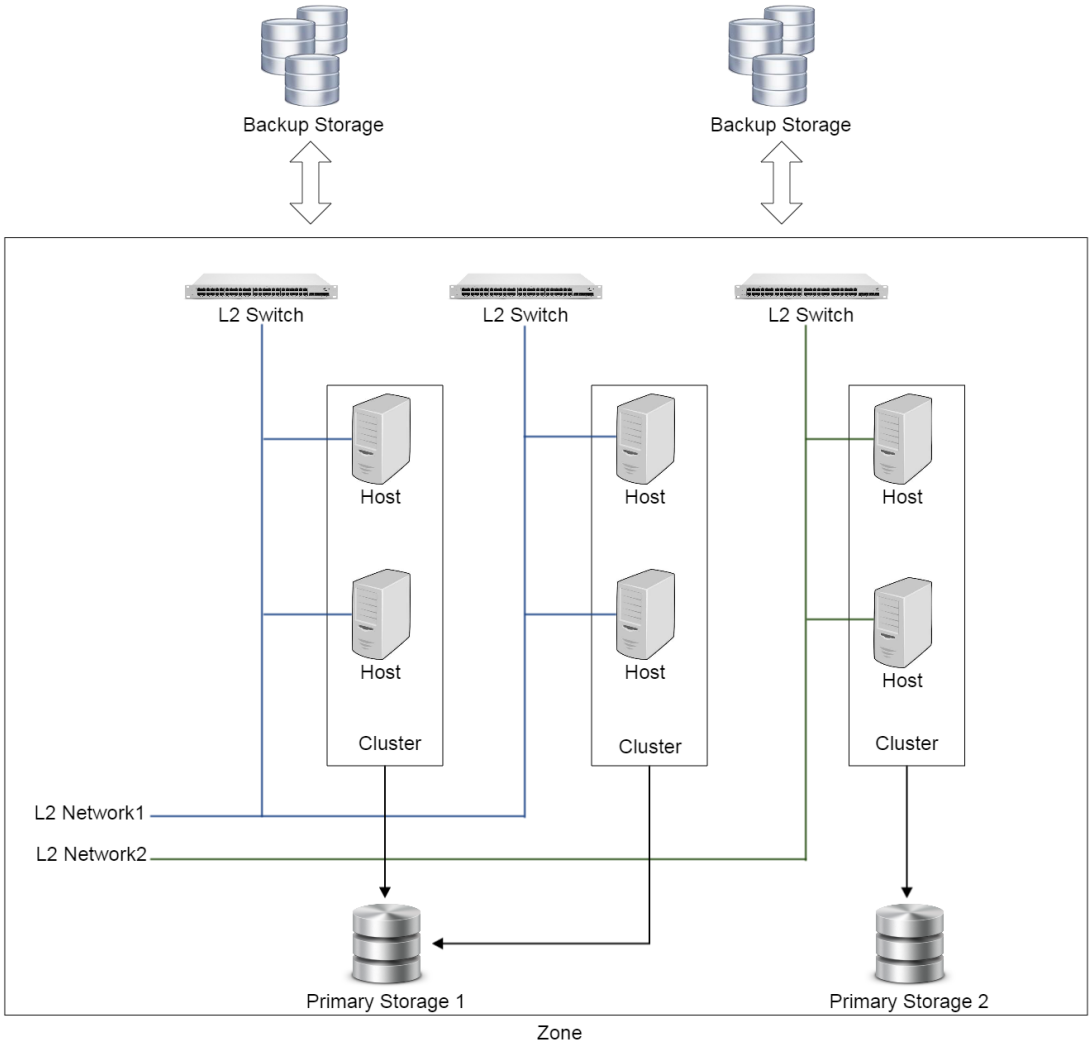
In this multiple nodes setup, the RabbitMQ server and MySQL database server are moved out to dedicated Linux machines; ZStack management nodes and Ansible are installed on every Linux server; multiple management nodes share the same RabbitMQ message server and MySQL database. ZStack UI servers, which also send API requests to management nodes through RabbitMQ, are deployed behind a load balancer which dispatches requests from users.

In terms of clustering RabbitMQ and MySQL, admin can setup two RabbitMQ servers and an additional slave MySQL database server.

ZStack's World View of A Cloud

IaaS software usually use some terms such as 'zone', 'cluster' to describe how facilities in a data center make up a cloud, so does ZStack. To reduce the learning curve and to eliminate misunderstandings caused by self-created terms, ZStack tries to use terminologies that have been well known in existing IaaS software and datacenters as much as possible.

Below is a diagram that how ZStack maps facilities of datacenters into its own language.



A datacenter, in ZStack's terms, is organized as follows:

- **Zone**:

  A zone is a logic group of resources, such as clusters, L2 networks, primary storage. ZStacks uses zones to define visibility boundary between resources. For example, a primary storage in zone A is not visible to a cluster in zone B. In practice, zones can also be used as isolated domains for fault tolerance, just as Amazon EC2 availability zones.

- **Cluster**:

  A cluster is a logic group of hosts. Hosts in the same cluster must have the same operating systems(hypervisor) and network configurations. Clusters are also known as host aggregations or host pools in other IaaS software.

- **Host**:

  A host is a physical server installed with an operating system(hypervisor) to run VMs.

- **L2 Network**:

  A L2 network is an abstraction of a layer 2 broadcast domain. Any technology, as long as providing a layer 2 broadcast domain, can be a type of L2 Network in ZStack. For example, VLAN, VxLan, or SDN technologies that create layer 2 overlay on layer 3 network.

- **Primary Storage**:

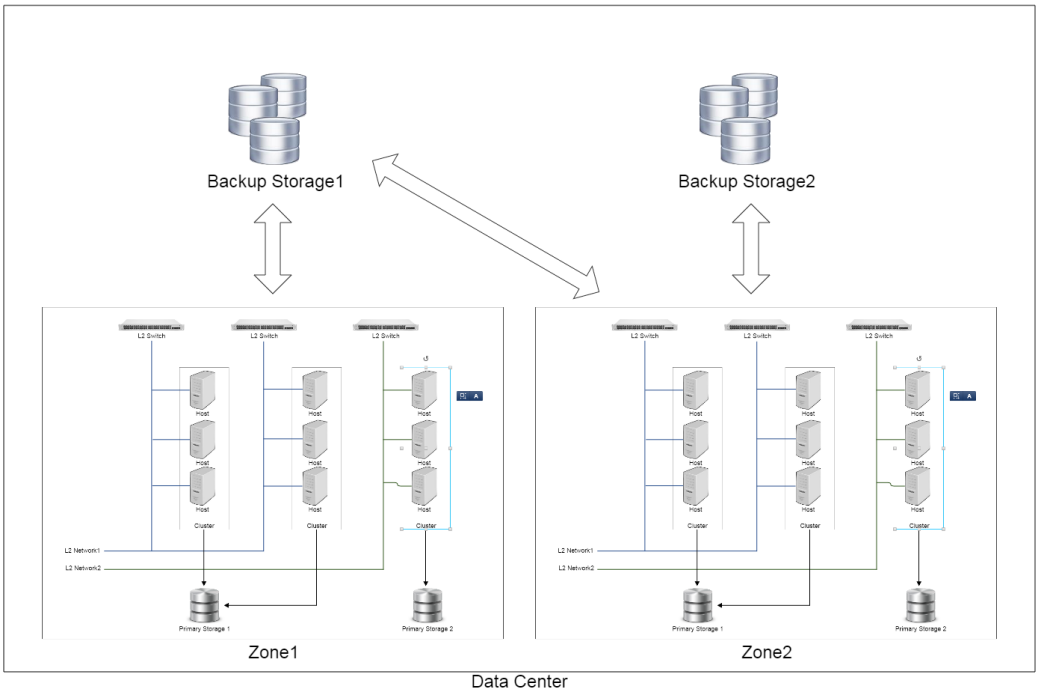
  A primary storage provides disk spaces to store VMs' volumes which will be accessed by VMs' operating system during running. Primary Storage can be either filesystem based like NFS or block storage based like ISCSI.

- **Backup Storage**:

  A backup Storage provides disk spaces to store images and volume snapshots both of which can be used to create volumes. Files on backup storage are not directly accessible to VMs; before being used, they need to be downloaded to primary storage. Backup Storage can either be filesystem based or object storage based.

ZStack uses a so-called 'attaching strategy' to describe relationships between resources, for example, a cluster can be attached with multiple primary storage and L2 networks, vice versa. See related chapters(e.g. primary storage, L2 network) for details.

A data center can have one or more zones. A diagram of multiple zones looks like:



Note: For simplicity, the diagram omits some facilities like aggregation switches, core switches, routers, load bal-

ancer, firewalls and so on.

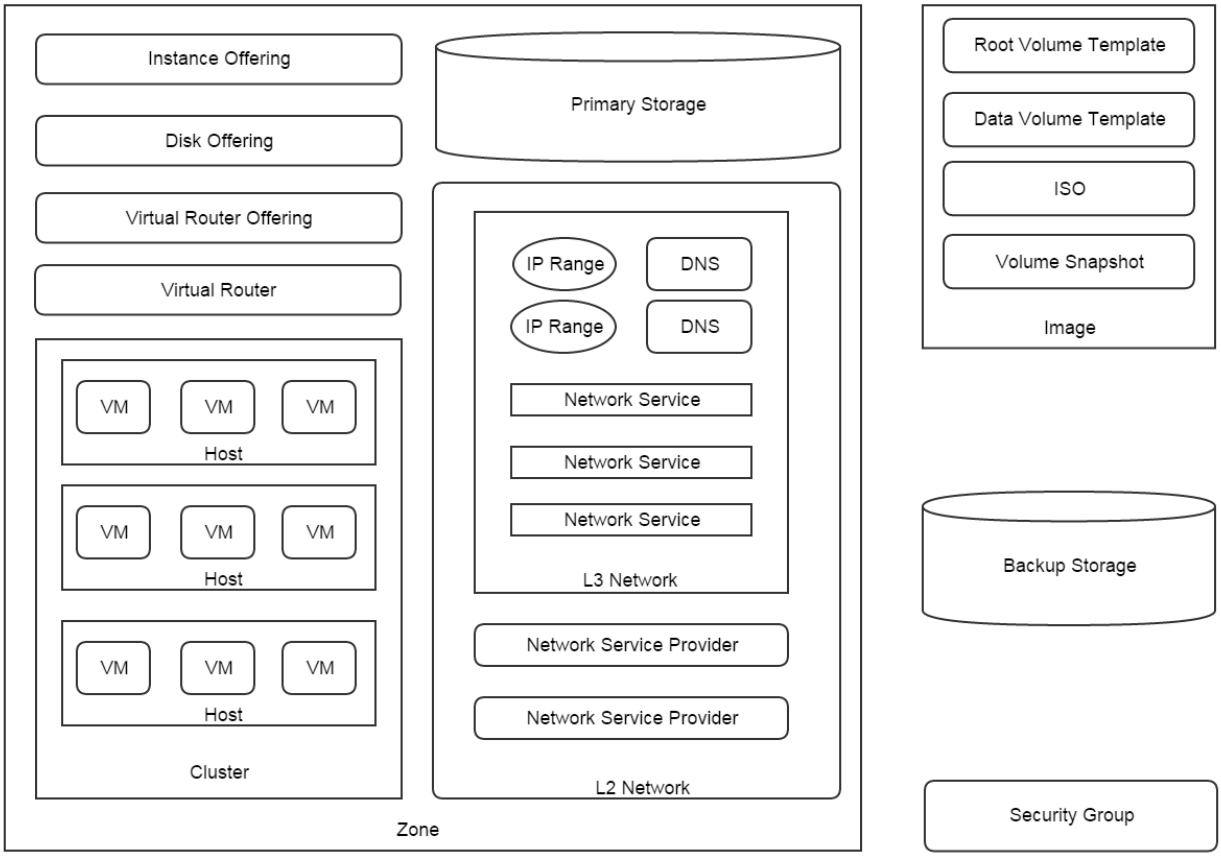
Besides above terms describing datacenter facilities, there are some other terms such as VM, instance offering, disk offering, which describe virtual resources; check details in relevant chapters.

2.2 Resource Model

Table of contents

- *Resource Model*
  - *Resource Relationship*
  - *Resource Properties*
  - *Resource Operations*
    - *Create Resources*
    - *Read Resources*
    - *Update Resources*
    - *Delete Resources*

ZStack is essentially a configuration management system for resources in the cloud. Resources can be physical resources(e.g. hosts, networks, storage) or virtual resources(e.g. VMs). In this version, a full diagram of ZStack resources is like:



Note: This diagram aims to give an overall idea that what ZStack resources look like. It neither exhibits exact

relationship among resources nor shows amount of resources.

2.2.1 Resource Relationship

Resources have four relationships:

- **Parent - Child**:

  A resource can be the parent or a child of another resource. For example, a cluster is a child resource of a zone, while a zone is the parent resource of a cluster.

- **Ancestor - Descendant**:

  A resource can be the lineal ancestor or a lineal descendant of another resource. For example, a zone is the ancestor resource of VMs; a VM is a descendant resource of a zone.

- **Sibling**:

  Resources sharing the same parent resource are siblings. For example, clusters, hosts, primary storage, L2 networks are sibling resources because all of them are child resources of zones.

- **Friend**:

  Resources which don't have above three relationships but still need to cooperate with each other in some scenarios are friends. For example, primary storage and backup storage are friends, because primary storage need to download images from backup storage in order to create VMs.

Resources that don't have any relationship are irrelevant resources; for example, security groups and clusters are irrelevant resources.

Note: In this version, ZStack doesn't have a concept of region, so zones and backup storage doesn't have a parent resource; however, they are still considered as siblings.

2.2.2 Resource Properties

There are four properties common to almost all resources:

- **UUID**:

  ZStack uses UUIDv4 (Universally Unique Identifier) to uniquely identify a resource. Unlike regular UUIDs which has four hyphens in string, the UUIDs ZStack use have hyphens stripped. For example, 80b5ca2c76154da298a1a248b975372a.

- **Name**:

  Names are human readable strings with maximum 255 characters. Names can be duplicated, as ZStack doesn't use them as resource identifiers. Names can have any visible ASCII characters(e.g. %, #, ^, space); however, putting symbols in a name may make query APIs hard to use. The best practice for naming a resource is only using letters, digits, '-'(hyphen), and '_'(underscore).

  Note: Please avoid using ','(comma) in names, though it's legal. In query APIs, ZStack uses comma to split a value into a list when the condition operator is '?='(which means 'in'). For example, querying VMs by a condition 'name' and an operator '?=' is like:

```
QueryVmInstance name?=vm1,vm2,vm3
```

what it does is: finding out VMs whose names are vm1 or vm2 or vm3. For people familiar with SQL, it is equal to:

```
select * from VmInstance where name in ('vm1', 'vm2', 'vm3')
```

if you have a comma in VMs' names, for example, 'v,m1', then the query becomes:

```
QueryVmInstance name?=v,m1,vm2,vm3
```

which turns out to find VMs whose names are in ['v', 'm1', 'vm2', 'vm3]

- **Description**:

  Descriptions are human readable strings with maximum 2048 characters. Still, ZStack doesn't enforces any limited character set on descriptions.

- **Created Date**:

  A immutable date indicating the time that resources were created.

- **Last Operation Date**:

  A date indicating the last time resources were updated. The date changes every time after a update has been performed on a resource; updates can be either from user operations or ZStack's internal operations.

Note: Some resources may not have names and descriptions, for example, DNS, security group rules. These resources are not considered as independent resources and must be with their parent resources.

Each resource may have its specific properties, for example, VMs have a property 'hostUuid'. As ZStack uses JSON in APIs, properties of resources are encompassed in a JSON map in most API responses. The JSON map is called *'inventory'* in ZStack's language. In following chapters, when talking about an inventory of a resource, we are referring to the map containing the resource's properties. Here is an example of VM inventory:


```
{
  "inventory": {
    "uuid": "94d991c631674b16be65bfdf28b9e84a",
    "name": "TestVm",
    "description": "Test",
    "zoneUuid": "acadddc85a604db4b1b7358605cd6015",
    "clusterUuid": "f6cd5db05a0d49d8b12721e0bf721b4c",
    "imageUuid": "061141410a0449b6919b50e90d68b7cd",
    "hostUuid": "908131845d284d7f821a74362fff3d19",
    "lastHostUuid": "908131845d284d7f821a74362fff3d19",
    "instanceOfferingUuid": "91cb47f1416748afa7e0d34f4d0731ef",
    "rootVolumeUuid": "19aa7ec504a247d89b511b322ffa483c",
    "type": "UserVm",
    "hypervisorType": "KVM",
    "createDate": "Jun 1, 2015 6:11:47 PM",
    "lastOpDate": "Jun 1, 2015 6:11:47 PM",
    "state": "Running",
    "vmNics": [
      {
        "uuid": "6b58e6b2ba174ef4bce8a549de9560e8",
        "vmInstanceUuid": "94d991c631674b16be65bfdf28b9e84a",
        "usedIpUuid": "4ecc80a2d1d93d48b32680827542ddbb",
        "l3NetworkUuid": "55f85b8fa9a647f1be251787c66550ee",
        "ip": "10.12.140.148",
        "mac": "fa:f0:08:8c:20:00",
```

```
        "netmask": "255.0.0.0",
        "gateway": "10.10.2.1",
        "deviceId": 0,
        "createDate": "Jun 1, 2015 6:11:47 PM",
        "lastOpDate": "Jun 1, 2015 6:11:47 PM"
      },
      {
        "uuid": "889cfcab8c08409296c649611a4df50c",
        "vmInstanceUuid": "94d991c631674b16be65bfdf28b9e84a",
        "usedIpUuid": "8877537e11783ee0bfe8af0fcf7a6388",
        "l3NetworkUuid": "c6134efd3af94db7b2928ddc5deba540",
        "ip": "10.4.224.72",
        "mac": "fa:e3:87:b1:71:01",
        "netmask": "255.0.0.0",
        "gateway": "10.0.0.1",
        "deviceId": 1,
        "createDate": "Jun 1, 2015 6:11:47 PM",
        "lastOpDate": "Jun 1, 2015 6:11:47 PM"
      },
      {
        "uuid": "cba0da7a12d44b2e878dd5803d078337",
        "vmInstanceUuid": "94d991c631674b16be65bfdf28b9e84a",
        "usedIpUuid": "f90d01a098303956823ced02438ae3ab",
        "l3NetworkUuid": "c7e9e14f2af742c29c3e25d58f16a45f",
        "ip": "10.29.42.155",
        "mac": "fa:2d:31:08:da:02",
        "netmask": "255.0.0.0",
        "gateway": "10.20.3.1",
        "deviceId": 2,
        "createDate": "Jun 1, 2015 6:11:47 PM",
        "lastOpDate": "Jun 1, 2015 6:11:47 PM"
      }
    ],
    "allVolumes": [
      {
        "uuid": "19aa7ec504a247d89b511b322ffa483c",
        "name": "ROOT-for-TestVm",
        "description": "Root volume for VM[uuid:94d991c631674b16be65bfdf28b9e84a]",
        "primaryStorageUuid": "24931b95b45e41fb8e41a640302d4c00",
        "vmInstanceUuid": "94d991c631674b16be65bfdf28b9e84a",
        "rootImageUuid": "061141410a0449b6919b50e90d68b7cd",
        "installUrl": "/opt/zstack/nfsprimarystorage/prim-24931b95b45e41fb8e41a640302d4c00/rootVolume
        "type": "Root",
        "format": "qcow2",
        "size": 3.221225472E10,
        "deviceId": 0,
        "state": "Enabled",
        "status": "Ready",
        "createDate": "Jun 1, 2015 6:11:47 PM",
        "lastOpDate": "Jun 1, 2015 6:11:47 PM"
      }
    ]
  }
}
```

2.2.3 Resource Operations

Resources support full or partial CRUD(Create, Read, Update, Delete) operations.

Create Resources

Every resource has own creational APIs. There is one parameter 'resourceUuid' common to all creational APIs. When 'resourceUuid' is not null, ZStack will use its value as the UUID for the resource being created; otherwise ZStack will automatically generate a UUID.

> **Warning:** When using 'resourceUuid', please make sure the UUID you provide is a UUIDv4 with hyphens striped. Otherwise, ZStack will return an invalid argument error if it's not a valid UUIDv4 with hyphens stripped, or an internal error if there has been a resource of the same type with the same UUID in the database.

Here is an example of creating a cluster:

```
CreateCluster name=cluster1 description='awesome cluster' hypervisorType=KVM zoneUuid=061141410a0449
```

or:

```
CreateCluster resourceUuid=f31e38309e2047beac588e111fa2051f name=cluster1 description='awesome cluste
```

Read Resources

Every resource has own query API that returns a list of inventories for read. For details, see *Query*. Here is an example of querying VMs:

```
QueryVmInstance allVolumes.type=Data allVolumes.size>1099511627776
```

The example does: finding out all VMs which have one or more data volumes with size greater than 1099511627776 bytes(1T)

Update Resources

A resource can be updated by various APIs. Updating a resource is actually performing an action to the resource. For example, starting a VM, stopping a VM. Please refer to corresponding chapters for actions for resources. Here is an example of starting a VM:

```
StartVmInstance uuid=94d991c631674b16be65bfdf28b9e84a
```

Most update APIs will return a resource inventory.

Delete Resources

A resource can be deleted. ZStack's philosophy for deleting is: every resource should be deletable; and deleting a resource should always be success unless user allows an expected failure; for example, a plugin may allow user to set a 'none-deletable' tag on a VM, and throw an error when the VM is being deleted.

Deleting a resource is not always easy in IaaS, especially for a resource that has many descendants; some software hard code to delete all descendant resources; some software simply throws an error when a resource being deleted still has descendant resources alive.

ZStack handles deleting in an elegant way. When a resource is being deleted, a so-called Cascade Framework will calculate relationships among this resource and rest resources in the cloud, and propagate proper actions to related

resources if necessary. For example, when deleting a zone, a deleting action will be spread to all descendants of the zone, which means all descendant resources like VMs, hosts, clusters, L2 Networks in this zone will be deleted before the zone deleted; and backup storage attached to the zone will be detached. With the cascade framework, deleting resources in ZStack is easy and reliable.

Every resource has own deleting API. A parameter *deleteMode* which has options **Permissive** and **Enforcing** is common to all deleting APIs. When *deleteMode* is set to Permissive, ZStack will stop the deleting if an error happens, or the deleting is not permitted; in this case, an error code with detailed reason will be returned. When *deleteMode* is set to Enforcing, ZStack will ignore any errors and permissions but delete resources directly; in this case, a deleting will always be success.

Here is an example of deleting a VM:

```
DestroyVmInstance uuid=94d991c631674b16be65bfdf28b9e84a
```

2.3 Command Line Tool

Table of contents

- *Command Line Tool*
  - *Overview*
  - *Usage*
    - *Connect to ZStack management node*
    - *Modes*
    - *LogIn*
    - *LogOut*
    - *Execute API Commands*
    - *View Command History*
    - *Export Command History*

2.3.1 Overview

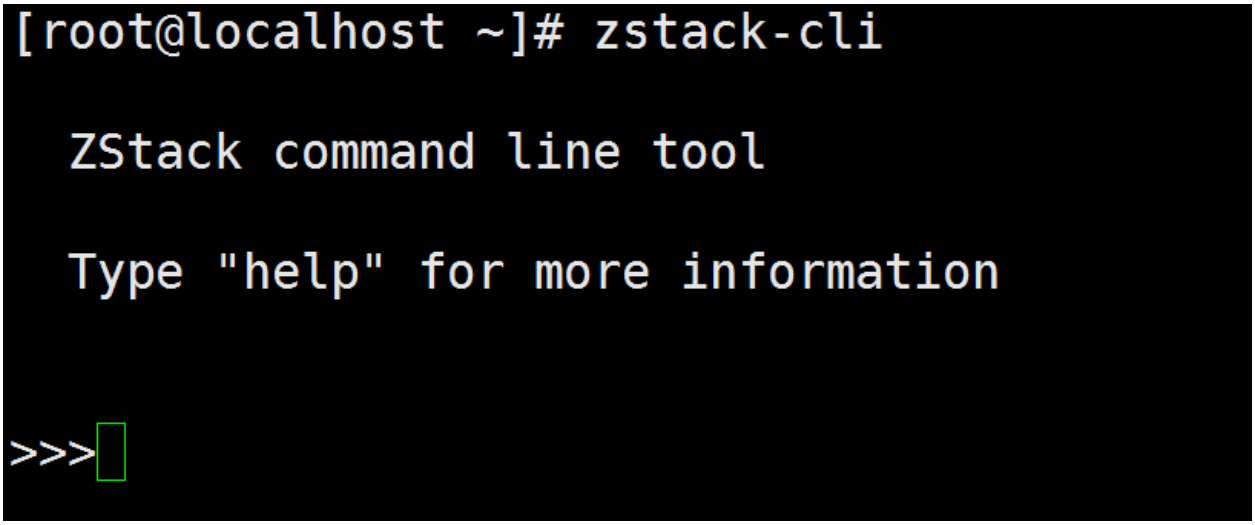
zstack-cli is the command line tool for users to execute all ZStack APIs. All API examples in this user manual are demonstrated using zstack-cli.

As ZStack is built on SOA(Service Oriented Architecture), all ZStack APIs are essentially messages; for example, you will see a CLI command called *StartVmInstance* in VM related chapter, which is actually mapping to the API message: APIStartVmInstanceMsg; nevertheless, people are more familiar with HTTP calls than messages, so ZStack ships a builtin HTTP server that wraps all API messages into HTTP post requests. zstack-cli is built on calling APIs through the builtin HTTP server.

2.3.2 Usage

Connect to ZStack management node

zstack-cli is installed by default after you install a ZStack management node. You can launch it by simply typing 'zstack-cli' in a shell console:



if no parameters are provided, zstack-cli will connect to 8080 port on localhost; to connect a remote ZStack management node, you can use options '-s' and '-p' to specify IP and port:

```
[root@localhost ~]# zstack-cli --help
Usage: -c [options]

Options:
  -h, --help  show this help message and exit
  -s HOST     [Optional] IP address or DNS name of ZStack management node.
              Default value: localhost
  -p PORT     [Optional] Port that ZStack management node is listening on.
              Default value: 8080
[root@localhost ~]# zstack-cli -s 192.168.0.212

  ZStack command line tool

  Type "help" for more information


>>>
```

Note: ZStack management nodes are running in Java servlet containers, for example, Tomcat, whose port numbers are rarely changed; most of the time you only need to specify the IP by '-s'.

if you have a multi-node deployment, you can connect the zstack-cli to any management nodes.

Modes

zstack-cli can work in a command mode that receives parameters from shell, runs once, and prints results to the shell console, for example:

```
[root@localhost ~]# zstack-cli -s 192.168.0.212 QueryZone name=test
2015-05-02 18:01:53,280 DEBUG [apibinding.api] async call[url: http://192.168.0.212:8080/zstack/api
/, request: {"org.zstack.header.zone.APIQueryZoneMsg": {"session": {"uuid": "564d907e58ae42fcb8a95e
6024d9b9e5"}, "conditions": [{"name": "name", "value": "test", "op": "="}]}}]
{
    "inventories": [
        {
            "createDate": "May 2, 2015 6:02:16 PM",
            "lastOpDate": "May 2, 2015 6:02:16 PM",
            "name": "test",
            "state": "Enabled",
            "type": "zstack",
            "uuid": "18107d51765f49a2ac0ec434e58ff5bb"
        }
    ],
    "success": true
}

Time costing: 0.258782s
[root@localhost ~]#
```

or an interactive shell mode that keeps a session for continuously executing, for example:

```
  Type "help" for more information


>>>query
GenerateInventoryQueryDetails     GenerateQueryableFields
QueryApplianceVm                  QueryBackupStorage
QueryCluster                      QueryDiskOffering
QueryEip                          QueryGlobalConfig
QueryHost                         QueryImage
QueryInstanceOffering             QueryIpRange
QueryL2Network                    QueryL2VlanNetwork
QueryL3Network                    QueryManagementNode
QueryNetworkServiceL3NetworkRef   QueryNetworkServiceProvider
QueryPortForwardingRule           QueryPrimaryStorage
QuerySecurityGroup                QuerySecurityGroupRule
QuerySftpBackupStorage            QuerySystemTag
QueryTag                          QueryUserTag
QueryVip                          QueryVirtualRouterOffering
QueryVmInstance                   QueryVmNic
QueryVmNicInSecurityGroup         QueryVolume
QueryVolumeSnapshot               QueryVolumeSnapshotTree
QueryZone

>>>query
```

people usually prefer interactive mode for manual execution but command mode for script integration.

LogIn

In this ZStack version(0.6), the IAM(Identity and Access Management) system is not ready; only one account 'admin' with default password('password') is available. Before executing any commands, you need to run the login command 'LogInByAccount' to get a session token which is automatically saved by zstack-cli to ~/.zstack/cli/session and you

don't need to keep it separately:

```
>>> LogInByAccount accountName=admin password=password
```

LogOut

Once you finish your work, you can use 'LogOut' to invalidate current session:

```
>>> LogOut
```

the LogOut command receives a parameter 'sessionUuid', but you don't need to provide it as zstack-cli will retrieve it from where it's kept.

Execute API Commands

Every API is a command with several parameters, you can execute them in either command mode or interactive mode:

```
>>> StartVmInstance uuid=11be8ac6adad44c68ae02493cba29846
```

```
[root@localhost ~]# zstack-cli StartVmInstance uuid=11be8ac6adad44c68ae02493cba29846
```

Note: In interactive mode, you can use Tab key to auto-complete a command or remind you about candidate parameters.

View Command History

You can use 'more' command to view your command history, for example:

```
>>> more
```

or:

```
[root@localhost ~]# zstack-cli more
```

the result format is the same to Linux *more* command, you can scroll up/down and search.

```
[NUM]   COMMAND
-----------------------------------------------
[1]      QueryVmInstance
[2]      LogOut
[3]      QueryZone state=Enabled
[4]      QueryVmInstance state=Running
[5]      QueryVmInstance
[6]      LogInByAccount accountName=admin password=password
Usage:
       >>>more NUM      #show the No. NUM Command result
       >>>more          #show all available NUM and Command. The failure command will be marked wi
th "!" before it.
(END)
```

to view the details of a command, use 'more' command following a command number:

```
>>> more 6
```

or:

```
[root@localhost ~]# zstack-cli more 6
```

the result is like:

```
Command:
        LogInByAccount accountName=admin password=password
Result:
{
    "inventory": {
        "uuid": "11be8ac6adad44c68ae02493cba29846"
    },
    "success": true
}
(END)
```

Note: Viewing command details is very useful when output of a command is larger than the screen size; for example, the result of QueryVmInstance.

Export Command History

You can export command history by 'save' command, saving one history each time or multiple histories at once:

```
>>> save 1
Saved command: 1 result to file: /home/root/QueryZone-1.json
```

```
[root@localhost ~]# zstack-cli -s 192.168.0.212 save 1
Saved command: 1 result to file: /home/root/QueryZone-1.json
```

or:

```
>>>save 1,2,3
Saved command: 1 result to file: /home/root/QueryZone-1.json
Saved command: 2 result to file: /home/root/CreateZone-2.json
Saved command: 3 result to file: /home/root/LogInByAccount-3.json
```

```
[root@localhost ~]# zstack-cli -s 192.168.0.212 save 1,2,3
Saved command: 1 result to file: /home/root/QueryZone-1.json
Saved command: 2 result to file: /home/root/CreateZone-2.json
Saved command: 3 result to file: /home/root/LogInByAccount-3.json
```

by default results are saved to current working folder, you can specify a destination folder by supplying an extra parameter of folder path:

```
>>> save 1 /tmp
save history command 1 result to /tmp/COMMAND-1.json
```

2.4 Query

Table of contents

- *Query*
  - *Overview*
  - *Architecture*
    - *Query API Parameters*
    - *Query Condition*
    - *CLI Query Conditions*
    - *Join(Expanded Query)*
    - *Query List*
    - *Query Tags*
    - *Avoid Loop Query*
    - *Use Query Efficiently*
  - *Examples*
    - *Normal Query*
    - *Query Count*
    - *Normal Query With Count*
    - *Set Limit*
    - *Set Start*
    - *Select Fields*
    - *Sort*

2.4.1 Overview

A main challenge for users operating large clouds is to find wanted resources accurately and quickly. For example, to find a VM which has an EIP (17.12.53.8) out of 100,000 VMs. ZStack provides comprehensive APIs that can query every field of every resource. See The Query API for the architecture design.

2.4.2 Architecture

Every ZStack resource groups its properties as an inventory in JSON format; for example, a zone inventory:

```
{
  "uuid": "b729da71b1c7412781d5de22229d5e17",
  "name": "TestZone",
  "description": "Test",
  "state": "Enabled",
  "type": "zstack",
  "createDate": "Jun 1, 2015 6:04:52 PM",
  "lastOpDate": "Jun 1, 2015 6:04:52 PM"
}
```

a resource inventory can include inventories of other resources; for example, a L3 network inventory contains IP range inventories:

```
{
    "createDate": "Nov 10, 2015 7:52:57 PM",
    "dns": [
        "8.8.8.8"
    ],
```

```
    "ipRanges": [
        {
            "createDate": "Nov 10, 2015 7:52:58 PM",
            "endIp": "192.168.0.190",
            "gateway": "192.168.0.1",
            "l3NetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
            "lastOpDate": "Nov 10, 2015 7:52:58 PM",
            "name": "ipr-mmbj",
            "netmask": "255.255.255.0",
            "startIp": "192.168.0.180",
            "uuid": "13238c8e0591444e9160df4d3636be82"
        }
    ],
    "l2NetworkUuid": "33107835aee84c449ac04c9622892dec",
    "lastOpDate": "Nov 10, 2015 7:52:57 PM",
    "name": "L3-SYSTEM-PUBLIC",
    "networkServices": [],
    "state": "Enabled",
    "system": true,
    "type": "L3BasicNetwork",
    "uuid": "95dede673ddf41119cbd04bcb5d73660",
    "zoneUuid": "3a3ed8916c5c4d93ae46f8363f080284"
}
```


there are two types of inventory fields: primitive field and nested field; a field is of primitive types of number, string, boolean and date; in above example, uuid, name, system are primitive fields; a nested field is of composite types which usually represent inventories of other resources; in above example, ipRanges is a nested fields.

Note: A nested field can only be queried by its sub-fields; for example, for the field *ipRanges*, you cannot do:

```
QueryL3Network ipRanges='[{"name":"ipr-mmbj""}]'
```

instead, you need to query its sub-field:

```
QueryL3Network ipRanges.name=ipr-mmbj
```

Every field of every inventory is queryable unless it's explicitly stated as unqueryable; for an inventory, there is a corresponding query API, for example, QueryZone, QueryHost, QueryVmInstance; the responses of query APIs always carry a list of inventories, or an empty list if no matching result is found. A query response is like:


```
{
  "inventories": [
      {
          "availableCpuCapacity": 13504,
          "availableMemoryCapacity": 16824565760,
          "clusterUuid": "b429625fe2704a3e94d698ccc0fae4fb",
          "createDate": "Nov 10, 2015 6:32:43 PM",
          "hypervisorType": "KVM",
          "lastOpDate": "Nov 10, 2015 6:32:43 PM",
          "managementIp": "192.168.0.212",
          "name": "U1404-192.168.0.212",
          "state": "Enabled",
          "status": "Connected",
          "totalCpuCapacity": 14400,
          "totalMemoryCapacity": 16828235776,
          "uuid": "d07066c4de02404a948772e131139eb4",
          "zoneUuid": "3a3ed8916c5c4d93ae46f8363f080284"
```

```
        }
    ],
    "success": true
}
```

A query API consists of a list of query conditions and several helper parameters:

Query API Parameters

Name	Description	Optional	Choices	Since
conditions	a list of *QueryCondition*			0.6
limit	the maximum number of inventories returned by the query API; default to 1000	true		0.6
start	the first inventory to return; default to 0	true		0.6
count	if true, the query response will return only count of inventories; default to false		• true • false	0.6
replyWithCount	if true, the query response will return both inventories and count; default to false		• true • false	0.6
sortBy	the field by which the result inventories will be sorted. The field must be a primitive field	true		0.6
sortDirection	if 'sortBy' is not null, this field specifies the sorting direction; default to 'asc'		• asc • desc	0.6
fields	a list of primitive fields; when specified, the result inventory will contain only those fields.	true		0.6

Query Condition

Query APIs receive a list of query conditions which have properties as following:

Name	Description	Optional	Choices	Since
name	field name			0.6
op	comparison operator		• = • != • >= • < • <= • in • not in • is null • is not null • like • not like	0.6
value	query value			0.6

a field name can be of a primitive field, or of a sub-field of a nested field, or of a sub-field of an expanded field(see *Join*); 'op' are comparison operators which are from SQL language.

Note: for CLI tool, some operators have different forms from SQL, listed in column 'CLI Form'

Op	CLI Form	Description
=	=	equal operator; case insensitive for string comparison
!=	!=	not equal operator; case insensitive for string comparison
>	>	greater than operator; check MySQL specification for string comparison
>=	>=	greater than or equal operator; check MySQL specification for string comparison
<	<	less than; check MySQL specification for string comparison
<=	<=	less than or equal operator; check MySQL specification for string comparison
in	?=	check whether a value is within a set of values
not in	!?=	check whether a value is NOT within a set of values
is null	=null	NULL value test
is not null	!=null	NOT NULL value test
like	~=	simple pattern matching. Use % to match any number of characters, even zero characters; use _ to matches exactly one character
not like	!~=	negation of simple pattern matching. Use % to match any number of characters, even zero characters; use _ to matches exactly one character

The relation among conditions is logical AND, it's the only relation supported in this ZStack version. For example:

```
QueryL3Network ipRanges.name=range1 name=L3Network1
```

is to find L3 networks whose names are 'L3Network1' AND which have one or more IP ranges with names 'range1'.

CLI Query Conditions

There are two ways to write conditions in CLI, one is the original form of query API:

```
QueryHost conditions='[{"name":"name", "op":"=", "value":"KVM1"}]'
```

another is CLI form:

```
QueryHost name=KVM1
```

I am sure you will prefer the CLI form as it's more intuitive and human readable. The CLI form always expresses query conditions in formula of:

```
condition_name(no_space)CLI_comparison_operator(no_space)condition_value
```

Warning: please note there is no space between condition_name and CLI_comparison_operator and condition_value:
`name=KVM1`
is valid but:
`name = KVM1`
is INVALID. See *CLI* for more details.

When typing in CLI, you can use *Tab* key for auto-completion and reminding you about queryable fields including primitive fields, nested fields, and expanded fields:

```
>>>QueryVmInstance
allVolumes.           cluster.              clusterUuid=          count=                createDate=           defaultL3NetworkUuid=  description=          fields=
host.                 hostUuid=             hypervisorType=       image.                imageUuid=            instanceOffering.      instanceOfferingUuid=  internalId=
lastHostUuid=         lastOpDate=           limit=                name=                 replyWithCount=       rootVolume.            rootVolumeUuid=        sortBy=
sortDirection=        start=                state=                timeout=              type=                 uuid=                  vmNics.                zone.
zoneUuid=
```

Join(Expanded Query)

Join is called expanded query in ZStack; it allows users to query a resource by fields that are neither primitive nor nested but other resources' fields that have relation to this resource; those fields are called expanded fields in ZStack's terms.

For example, to find the parent L3 network of a VM nic having an EIP with VIP 17.16.0.53:

```
QueryL3Network vmNic.eip.vipIp=17.16.0.53
```

here L3 network inventory has no field called 'vmNic.eip.vipIp'; however, it has a relation to VM nic inventory that has a relation to EIP inventory; so we can construct an expanded query that spans to three inventories: L3 network inventory, VM nic inventory, and EIP inventory. Thanks for this nuclear weapon, ZStack has around four millions query conditions and countless combinations of conditions. Let's see a more complex and artificial example:

```
QueryVolumeSnapshot volume.vmInstance.vmNics.l3Network.l2Network.attachedClusterUuids?=13238c8e059144
```

This complex query is to find volume snapshots created from volumes of VMs that have nics on L3 networks whose parent L2 networks are attached to a cluster of uuid equal to 13238c8e0591444e9160df4d3636be82. Though users will barely do such a query, it shows the power of the query APIs.

Note: Check query operations in each chapter for expanded queries a resource can make, or use CLI auto-completion as a reminder.

Query List

When a field is a list, it can contain primitive types such as int, long, string or nested inventories. Querying list has nothing special; we have this section to remind you that don't incorrectly think you can only use 'in'(?=) and 'not in'(!?=) when querying a list field; in fact, you can use all comparison operators; for example:

```
QueryL3Network dns~=72.72.72.%
```

is to find all L3 networks that have DNS like 72.72.72.*:

```
QueryL3Network ipRanges.startIp=192.168.0.10
```

is to find all L3 networks whose IP ranges starting with IP 192.168.0.10.

Query Tags

In section *tags* you will see every resource can have user tags and system tags both of which can be a part of query conditions. ZStack uses two special fields: *__userTag__* and *__systemTag__* for query; for example:

```
QueryVmInstance __userTag__?=web-tier-VMs
```

```
QueryHost __systemTag__?=os::distribution::Ubuntu managementIp=192.168.0.212
```

operators >, >=, <, <= only return resources that have tags matching specified conditions; 'is not null' returns resources that have tags; 'is null' returns resources that have no tags; !=, 'not in', 'not like' return resources that have tags not matching conditions as well as resources that have no tags.

Note: If you want to make negative comparison operators(!=, 'not in', 'not like') not to return resources that have no tags, you can use them with 'not null'. For example:

```
QueryVmInstance __userTag__!=database  __userTag__!=null
```

is to find VMs that have user tags not equal to 'database'.

Avoid Loop Query

Most ZStack resources have bi-direction expanded queries, for example, hosts have an expanded query to clusters and clusters also have an expanded query to hosts. This makes it's possible to query a resource from any directions, which may also lead to loop queries. For example:

```
QueryHost vmInstance.vmNics.eip.vmNic.vmInstance.uuid=d40e459b97db5a63dedaffcd05cfe3c2
```

is a loop query, it does the thing equal to:

```
QueryHost vmInstance.uuid=d40e459b97db5a63dedaffcd05cfe3c2
```

Behaviors of loop queries is undefined; you may or may not get the correct results. Please avoid loop query in your practice.

Use Query Efficiently

Query APIs are powerful that you can do the same thing by different routes. For example, to find VMs that are running on the host of UUID e497e90ab1e64db099eea93f998d525b, you can either do:

```
QueryVmInstance hostUuid=e497e90ab1e64db099eea93f998d525b
```

or:

```
QueryVmInstance host.uuid=e497e90ab1e64db099eea93f998d525b
```

The first one is more efficient, because it queries a primitive field which only involves the VM table; the later one is an expanded query which joins both VM table and host table. When your query condition is a UUID, it's always suggested querying the primitive field instead of the sub-field of an expanded field.

2.4.3 Examples

Normal Query

```
QueryL3Network name=L3-SYSTEM-PUBLIC
```

Query Count

```
QueryL3Network name=L3-SYSTEM-PUBLIC count=true
```

Normal Query With Count

```
QueryL3Network name=L3-SYSTEM-PUBLIC replyWithCount=true
```

Set Limit

```
QueryL3Network l2NetworkUuid=33107835aee84c449ac04c9622892dec limit=10
```

Set Start

```
QueryL3Network l2NetworkUuid=33107835aee84c449ac04c9622892dec start=10 limit=100
```

Note: Using start and limit, UI can implement pagination.

Select Fields

```
QueryL3Network fields=name,uuid l2NetworkUuid=33107835aee84c449ac04c9622892dec
```

Note: Only primitive fields can be selected.

Sort

```
QueryL3Network l2NetworkUuid=33107835aee84c449ac04c9622892dec sortBy=createDate sortDirection=desc
```

Note: Only primitive field can be used as sorted field.

2.5 Global Configurations

Table of contents

- *Global Configurations*
  - *Overview*
  - *Inventory*
    - *Example*
  - *Operations*
    - *Update Global Configurations*
  - *Other Configurations*
    - *statistics.on*
    - *node.heartbeatInterval*
    - *node.joinDelay*
    - *key.public*
    - *key.private*

2.5.1 Overview

Admins can use global configurations to configure a variety of features; all global configurations come with a default value; updating a global configuration doesn't require to restart the management node.

We arrange resource related global configurations in each chapter, for those configurations that don't specifically categorise in any resource we list them in this chapter.

2.5.2 Inventory

Name	Description	Optional	Choices	Since
category	configuration category			0.6
description	configuration description			0.6
name	configuration name			0.6
defaultValue	default value			0.6
value	current value			0.6

Example

```
{
    "category": "identity",
    "defaultValue": "500",
    "description": "Max number of sessions management server accepts. When this limit met, new sessio
    "name": "session.maxConcurrent",
    "value": "500"
}
```

2.5.3 Operations

Update Global Configurations

Admins can use UpdateGlobalConfig to update a global configuration. For example:

```
UpdateGlobalConfig category=host name=connection.autoReconnectOnError value=true
```

2.5.4 Other Configurations

For configurations that don't categorise in individual chapter.

statistics.on

Name	Category	Default Value	Choices
statistics.on	cloudBus	false	• true • false

Whether enables statistics that count time consuming of each message through JMX.

node.heartbeatInterval

Name	Category	Default Value	Choices
node.heartbeatInterval	managementServer	5	> 0

The interval that each management node writes heartbeat to database, in seconds.

node.joinDelay

Name	Category	Default Value	Choices
node.joinDelay	managementServer	0	>= 0

If non zero, each management node will delay random seconds from 0 to 'node.joinDelay' before publishing join event on the message bus. This avoid storm of join event when a large number of management nodes start at the same time.

key.public

Name	Category	Default Value	Choices
key.public	configuration	see your database	

ZStack will inject this public SSH key to Linux servers that need to deploy agents; in this version, the Linux servers include KVM host, virtual router VMs, SFTP backup storage. After injecting, ZStack will use *key.private* when needing SSH login.

key.private

Name	Category	Default Value	Choices
key.private	configuration	see your database	

The private SSH key ZStack uses to SSH login remote Linux servers; see *key.public*.

2.6 Tags

Table of contents

- *Tags*
  - *Overview*
  - *User Tags*
  - *System Tags*
  - *Name Convention*
  - *Resource Type*
  - *Operations*
    - *Create Tags*
      - *Parameters*
    - *Delete Tag*
      - *Parameters*
    - *Query Tags*

2.6.1 Overview

ZStack provides two types of tags to help users and plugins organize resources, introduce extra resource properties, and instruct ZStack to perform specific business logic. For the architecture design of tags see The Tag System.

2.6.2 User Tags

Users can create user tags on resources they own, which is particular useful when aggregating a set of similar resources; for example, users can put a tag 'web' on VMs that work as web servers:

```
CreateUserTag resourceType=VmInstanceVO resourceUuid=613af3fe005914c1643a15c36fd578c6 tag=web
```

```
CreateUserTag resourceType=VmInstanceVO resourceUuid=5eb55c39db015c1782c7d814900a9609 tag=web
```

```
CreateUserTag resourceType=VmInstanceVO resourceUuid=0cd1ef8c9b9e0ba82e0cc9cc17226a26 tag=web
```

and later on, use *Query API with tags* to retrieve those VMs:

```
QueryVmInstance __userTag__=web
```

Users can also use user tags cooperating with system tags to change ZStack's business logic; for example, users may want all VMs working as web servers to create their root volumes on a special primary storage which provides better IO performance by SSD; to do so, users can create a user tag 'forWebTierVM' on the primary storage:

```
CreateUserTag tag=forWebTierVM resourceType=PrimaryStorageVO resourceUuid=6572ce44c3f6422d8063b0fb262
```

then create a system tag on an instance offering:

```
CreateSystemTag tag=primaryStorage::allocator::userTag::forWebTierVM resourceType=InstanceOfferingVO
```

then, when users create a VM with the instance offering[uuid:8f69ef6c2c444cdf8c019fa0969d56a5], ZStack will make sure the VM's root volume will be created on only the primary storage with user tag 'forWebTierVM', in this case, which is the primary storage with UUID 6572ce44c3f6422d8063b0fb262cbc62.

2.6.3 System Tags

System tags have wider usage than user tags; users can use them to instruct ZStack to do some specific business logic, like the example in the section above. Plugins, which extend ZStack's functionality, can use system tags to introduce additional resource properties, or to record metadata which tightly bind to resources.

for example, to carry out live migration or live snapshot on KVM hosts, ZStack needs to know KVM hosts' libvirt version and QEMU version all of which are treated as meta data, so ZStack records them as system tags of hosts. For example, admins can view system tags of a KVM host by:

```
QuerySystemTag fields=tag resourceUuid=d07066c4de02404a948772e131139eb4
```

d07066c4de02404a948772e131139eb4 is the KVM host UUID, the output is like:

```
{
  "inventories": [
      {
          "tag": "capability:liveSnapshot"
      },
      {
          "tag": "qemu-img::version::2.0.0"
      },
      {
          "tag": "os::version::14.04"
      },
      {
          "tag": "libvirt::version::1.2.2"
      },
      {
          "tag": "os::release::trusty"
      },
      {
          "tag": "os::distribution::Ubuntu"
      }
  ],
  "success": true
}
```

this kind of system tags, which record meta data, are called inherent system tags; inherent system tags can only be created by ZStack's services or plugins, and cannot be deleted by DeleteTag API.

To add new functionality, a plugin usually needs to add new properties to a resource; though a plugin cannot change a resource's database schema to add a new column, it can create new properties as system tags of a resource. For example, when creating a VM, users can specify the VM's hostname for the default L3 network:

```
CreateVmInstance name=testTag systemTags=hostname::web-server-1 l3NetworkUuids=6572ce44c3f6422d8063b
```

this hostname is implemented by a system tag; if you look at *VM inventory in chapter 'Virtual Machine'*, there is no property called 'hostname'; however, you can find it from the VM's system tags:

```
QuerySystemTag fields=tag,uuid resourceUuid=76e119bf9e16461aaf3d1b47c645c7b7
```

```
{
  "inventories": [
      {
          "tag": "hostname::web-server-1",
          "uuid": "596070a6276746edbf0f54ef721f654e"
      }
  ],
```

```
  "success": true
}
```

this kind of system tags are non-inherent, users can delete them by DeleteTag; for example, if users want to change the hostname of the former VM to 'web-server-nginx', they can do:

```
DeleteTag uuid=596070a6276746edbf0f54ef721f654e
```

```
CreateSystemTag resourceType=VmInstanceVO tag=hostname::web-server-nginx resourceUuid=76e119bf9e1646
```

after stopping and starting the VM, the guest operating system will receive the new hostname as 'web-server-nginx'.

Note: System tags are pre-defined by ZStack's services and plugins; user cannot create a non-existing system tag on a resource. You can find resources' system tags in *Tags* section of every resource chapter.

2.6.4 Name Convention

Both user tags and system tags can have at most 2048 characters.

For user tags, there is no enforced name convention, but it's recommended to use human readable and meaningful strings.

For system tags, as defined by ZStack's services and plugins, they follow the format that uses *::* as delimiters.

2.6.5 Resource Type

When creating a tag, user must specify the resource type that the tag is associated with. In this version, a list of resource types is showed as follows:

ZoneVO
ClusterVO
HostVO
PrimaryStorageVO
BackupStorageVO
ImageVO
InstanceOfferingVO
DiskOfferingVO
VolumeVO
L2NetworkVO
L3NetworkVO
IpRangeVO
VipVO
EipVO
VmInstanceVO
VmNicVO
SecurityGroupRuleVO
SecurityGroupVO
PortForwardingRuleVO
VolumeSnapshotTreeVO
VolumeSnapshotVO

Derived resources use their parent types; for example, SftpBackupStorage's resourceType is 'BackupStorageVO'. In *Tags* section of every resource chapter, we will explain what resource types to use when creating tags.

2.6.6 Operations

Create Tags

There are two ways to create tags; for resources that have been created, users can use command CreateUserTag or CreateSystemTag to create a user tag or a system tag. For example:

```
CreateUserTag resourceType=DiskOfferingVO resourceUuid=50fcc61947f7494db69436ebbbefda34 tag=for-large
```

```
CreateSystemTag resourceType=HostVO resourceUuid=50fcc61947f7494db69436ebbbefda34 tag=reservedMemory
```

For a resource that is going to be created, as it's not been created yet, there is no resource UUID that can be referred in the CreateUserTag and CreateSystemTag commands; in this case, users can use *userTags* and *systemTags* fields, both of which are of a list type that receives a list of tags, of every *creational API command*; for example:

```
CreateVmInstance name=testTag systemTags=hostname::web-server-1
userTags=in-super-data-center,has-public-IP,hot-fix-applied-2015-5-1
l3NetworkUuids=6572ce44c3f6422d8063b0fb262cbc62
instanceOfferingUuid=04b5419ca3134885be90a48e372d3895 imageUuid=f1205825ec405cd3f2d259730d47d1d8
```

Parameters

CreateUserTag and CreateSystemTag have the same API parameters:

Name	Description	Optional	Since
resourceUuid	resource UUID; for example, VM's UUID uuid, instance offering's UUID		0.6
resourceType	resource type; see *resource type*		0.6
tag	tag string		0.6

Delete Tag

Users can use DeleteTag to delete a user tag or a non-inherent system tag. For example:

```
DeleteTag uuid=7813d03bb85840c489789f8df3a5915b
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	tag UUID			0.6

Query Tags

Users can use QueryUserTag to query user tags, for example:

```
QueryUserTag resourceUuid=0cd1ef8c9b9e0ba82e0cc9cc17226a26 tag~=web-server-%
```

or QuerySystemTag to query system tags, for example:

```
QuerySystemTag resourceUuid=50fcc61947f7494db69436ebbbefda34
```

Note: When querying tags, as the resourceUuid has uniquely identified a resource, you don't need to specify the resource type; for example:

```
QueryUserTag resourceUuid=0cd1ef8c9b9e0ba82e0cc9cc17226a26 resourceType=VmInstanceVO
```

is redundant because ZStack knows resourceUuid *0cd1ef8c9b9e0ba82e0cc9cc17226a26* maps to type *VmInstanceVO*.

And don't forget you can use *__userTag__* and *__systemTag__* to query resources with tags, see *Query API with tags*.

2.7 Zone

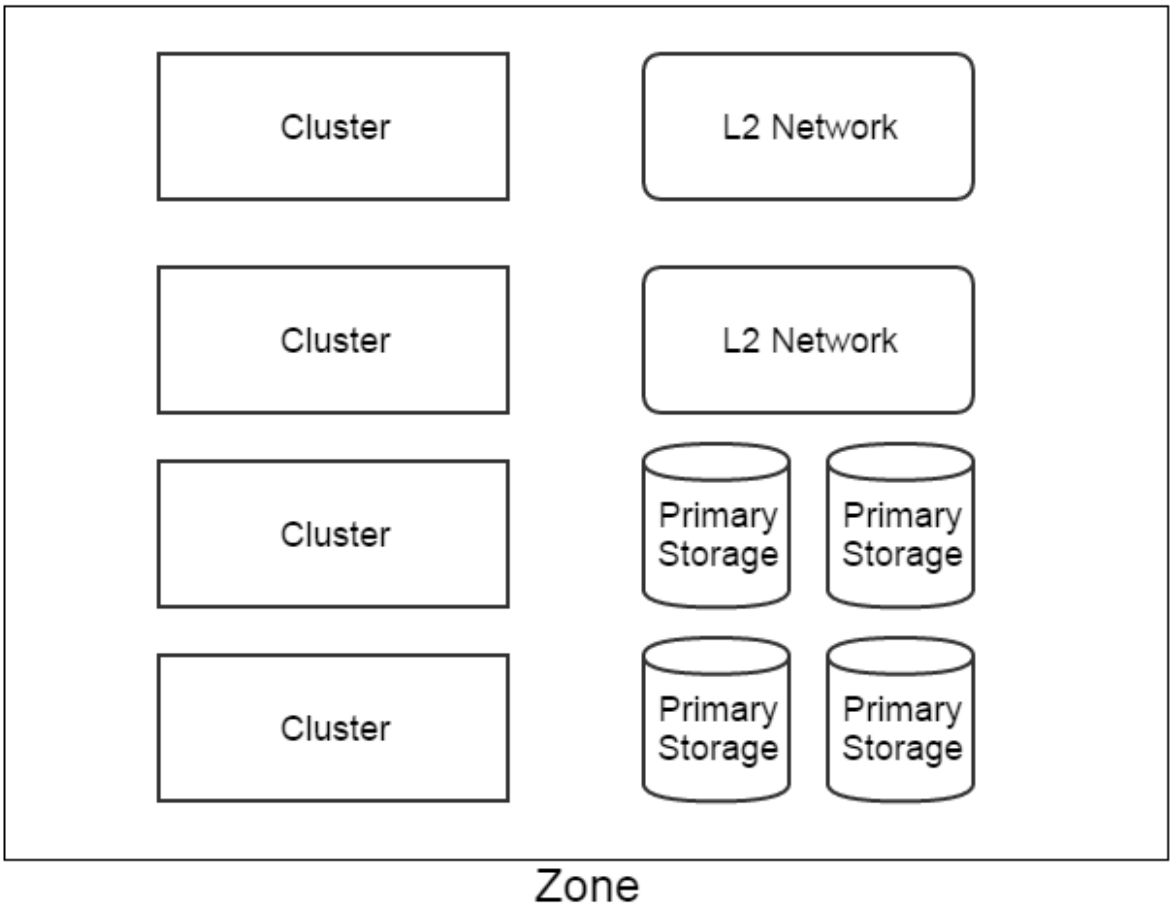
Table of contents

- *Zone*
  - *Overview*
  - *Inventory*
    - *Properties:*
    - *Example*
    - *State*
  - *Operations*
    - *Create Zone*
      - *Parameters*
    - *Delete Zone*
      - *Parameters*
    - *Change State*
      - *Parameters*
    - *Attach Backup Storage*
    - *Detach Backup Storage*
    - *Query Zone*
      - *Primitive Fields of Query*
      - *Nested And Expanded Fields of Query*
  - *Tags*
    - *System Tags*
      - *Reserved Capacity*

2.7.1 Overview

A zone is a logic group of resources such as primary storage, clusters, L2 networks; it defines a visibility boundary that resources in the same zone can see each other and establish relationships, while resources in different zones cannot. For example, a primary storage in zone A can be attached to a cluster also in zone A, but cannot be attached to a cluster in zone B.

Zones' child resources, including clusters, L2 Networks and primary Storage, are organized as follows:



Descendant resources of zones are not listed in above diagram. For instance, a host in a cluster is a descendant resource of the parent zone of the cluster.

As a logic resource, zones maps facilities in datacenters to logic groups. Though there is no enforcement on how facilities must be mapped, some advices are given to make things simple and clear:

- Hosts in the same physical layer 2 broadcast domain should be in the same zone, grouped as one or more clusters.
- Physical layer2 broadcast domains should not span multiple zones, and should be mapped as L2 networks in a single zone.
- Physical storage that provide disk spaces for VM volumes, known as primary storage, should not span multiple zones, and should be mapped as primary storage in a single zone.
- A datacenter can have multiple zones.

A zone can has one or more *Backup Storage* attached. Resources in a zone, for example primary storage, can only access backup storage attached to the zone. Also, a backup storage can be detached from a zone; after detaching, resources in the zone will not see the backup storage any more. Detaching backup storage is particularly useful when network typology changes in a datacenter, if the changes cause backup storage no longer accessible to resources of a zone.

2.7.2 Inventory

Properties:

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
state	see *zone state*		• Enabled • Disabled	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6
type	reserved field			0.6

Example

```
{
  "uuid": "b729da71b1c7412781d5de22229d5e17",
  "name": "TestZone",
  "description": "Test",
  "state": "Enabled",
  "type": "zstack",
  "createDate": "Jun 1, 2015 6:04:52 PM",
  "lastOpDate": "Jun 1, 2015 6:04:52 PM"
}
```

State

Zones have two states: Enabled and Disabled. When changing a zone's state, the operation will be cascaded to all clusters and hosts all of which belong to the zone. For example, disabling a zone will change states of all clusters and hosts in this zone to Disabled. Because no VM can be created or started on a disabled host, putting a zone into Disabled state can prevent any VM from being created or started in this zone.

Note: Admins can selectively enable hosts or clusters in a disabled zone or disable them in an enabled zone, in order to have fine-grained state control.

2.7.3 Operations

Create Zone

Admins can use CreateZone command to create a new zone. For example:

```
CreateZone name='San Jose Zone' description='this is a zone in San Jose datacenter'
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
type	reserved field, don't evaluate it	true		0.6
userTags	user tags, see *Create Tags*; resource type is ZoneVO	true		0.6
systemTags	system tags, see *Create Tags*; resource type is ZoneVO	true		0.6

Delete Zone

Admins can use DeleteZone command to delete a zone. For example:

```
DeleteZone uuid=28e94936284b45f99842ababfc3f976d
```

Danger: There is no way to recover a deleted zone.

Parameters

Name	Description	Optional	Choices	Since
uuid	zone uuid			0.6
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6

Change State

Admins can use ChangeZoneState command to change the state of a zone. For example:

```
ChangeZoneState stateEvent=enable uuid=737896724f2645de9372f11b13a48223
```

Parameters

Name	Description	Optional	Choices	Since
uuid	zone uuid			0.6
stateEvent	state trigger event. • enable: change state to Enabled • disable: change state to Disabled		• enable • disable	0.6

Attach Backup Storage

see *attach backup storage to zone*.

Detach Backup Storage

see *detach backup storage from zone*.

Query Zone

Admins can use QueryZone to query zones. For example:

```
QueryZone name=zone1
```

```
QueryZone vmInstance.uuid=13238c8e0591444e9160df4d3636be82
```

Primitive Fields of Query

see *zone inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
vmInstance	*vm inventory*	VMs belonging to this zone	0.6
cluster	*cluster inventory*	clusters belonging to this zone	0.6
host	*host inventory*	hosts belonging to this zone	0.6
primaryStorage	*primary storage inventory*	primary storage belonging to this zone	0.6
l2Network	*L2 network inventory*	L2 networks belonging to this zone	0.6
l3Network	*L3 network inventory*	L3 networks belonging to this zone	0.6
backupStorage	*backup storage inventory*	backup storage belonging to this zone	0.6

2.7.4 Tags

Admins can create user tags on a zone with resourceType=ZoneVO. For example:

```
CreateUserTag resourceType=ZoneVO resourceUuid=0cd1ef8c9b9e0ba82e0cc9cc17226a26 tag=privateZone
```

System Tags

Reserved Capacity

Tag	Description	Example	Since
host::reservedMemory::{capacity}	see *Host Capacity Reservation*	host::reservedMemory::1G	0.6

2.8 Cluster

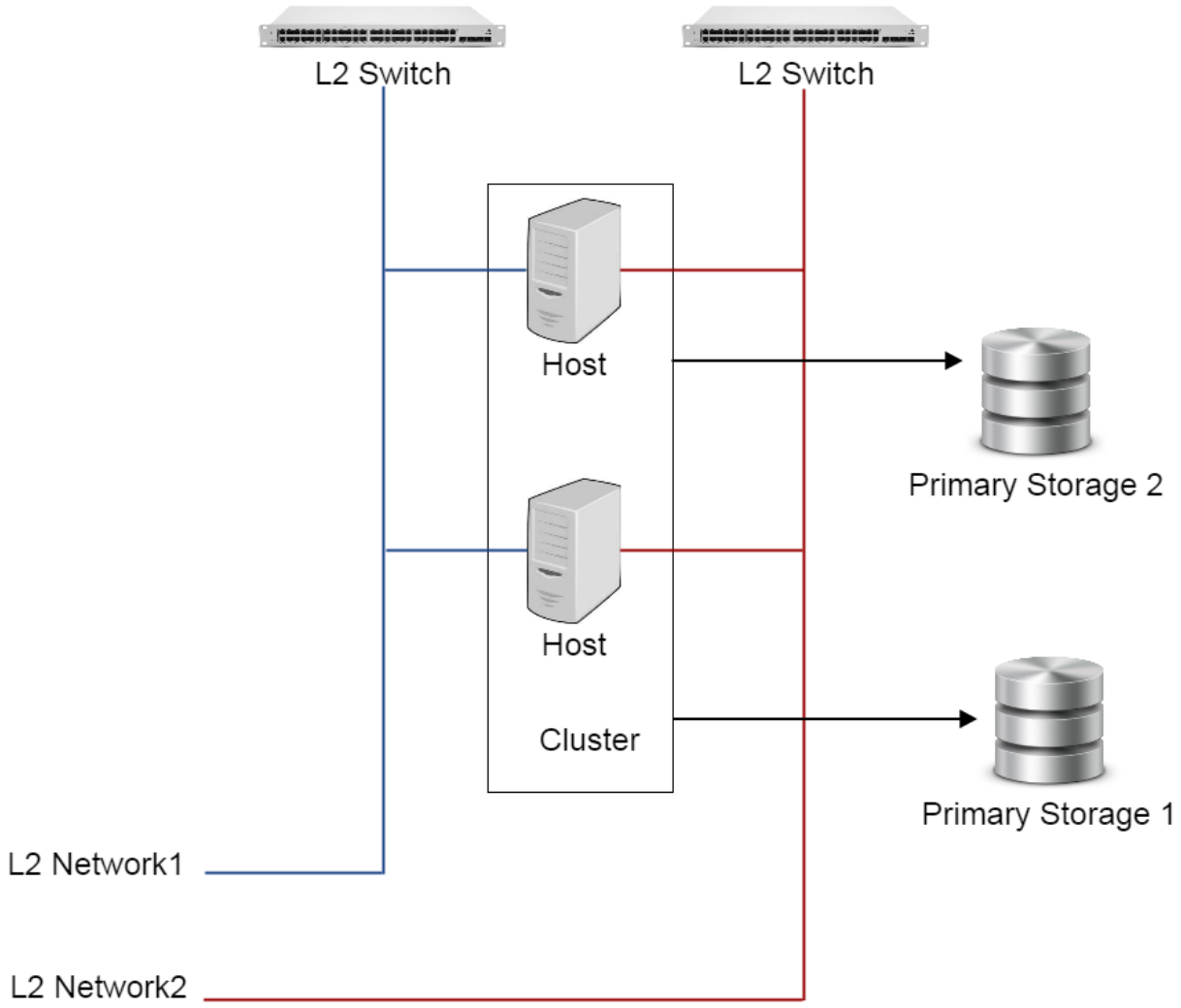
Table of contents

- *Cluster*
  - *Overview*
  - *Inventory*
    - *Properties*
    - *Example*
    - *Hypervisor Type*
    - *State*
  - *Operations*
    - *Create Cluster*
      - *Parameters*
    - *Delete Cluster*
      - *Parameters*
    - *Change State*
      - *Parameters*
    - *Attach Primary Storage*
      - *Parameters*
    - *Detach Primary Storage*
      - *Parameters*
    - *Attach L2 Network*
      - *Parameters*
    - *Detach L2 Network*
      - *Parameters*
    - *Query Cluster*
      - *Primitive Fields of Query*
      - *Nested And Expanded Fields of Query*
  - *Tags*
    - *System Tags*
      - *Reserved Capacity*

2.8.1 Overview

A cluster is a logic group of analogy hosts. Hosts in the same cluster must be installed with the same operating systems(hypervisor), have the same layer2 network connectivity, and can access the same primary storage. In real datacenters, a cluster usually maps to a rack.

A typical cluster and its relationship to primary storage, L2 networks is shown in below diagram.



A cluster can have one or more primary storage attached, as long as hosts in the cluster can all access these primary storage. Also, a primary storage can be detached from a cluster; this is particularly useful when network typology changes in datacenters, which causes the primary storage no longer accessible to hosts in the cluster.

A cluster can have one or more L2 networks attached, as long as hosts in the cluster are all in the physical layer2 broadcast domains those L2 networks represent. Also, a L2 network can be detached from a cluster, if network typology changes in the datacenter cause hosts in the cluster no longer in the layer2 broadcast domain of the L2 network.

The size of a cluster, which is the maximum hosts the cluster can contain, is not enforced.

2.8.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
hypervisorType	see *cluster hypervisor type*		• KVM	0.6
state	see *cluster state*		• Enabled • Disabled	0.6
zoneUuid	uuid of zone containing the cluster. See *zone*.			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6
type	reserved field			0.6
userTags	user tags, see *Create Tags*	true		0.6
systemTags	system tags, see *Create Tags*	true		0.6

Example

```
{
  "inventory": {
    "uuid": "c1bd173d5cd84f0e9e7c47195ae27ec6",
    "name": "cluster1",
    "description": "test",
    "state": "Enabled",
    "zoneUuid": "1b830f5bd1cb469b821b4b77babfdd6f"
    "hypervisorType": "KVM",
    "lastOpDate": "Jun 1, 2015 5:54:09 PM",
    "createDate": "Jun 1, 2015 5:54:09 PM",
    "type": "zstack",
  }
}
```

Hypervisor Type

Hypervisor type indicates what hypervisor(operating system) installed on hosts in the cluster. In this ZStack version, the only supported hypervisor is KVM.

State

Cluster has two states: Enabled and Disabled, just like *zone*. When changing the state of a cluster, the operation will be spread to all hosts of the cluster; For example, disabling a cluster will disable all hosts in the cluster as well.

Note: Admins can selectively enable hosts in a disabled cluster or disable them in an enabled cluster, in order to have fine-grained state control.

2.8.3 Operations

Create Cluster

Admins can use CreateCluster command to create a cluster. For example:

```
CreateCluster name=cluster1 hypervisorType=KVM zoneUuid=1b830f5bd1cb469b821b4b77babfdd6f
```

Parameters

Name	Description	Optional	Choices	Since
zoneUuid	uuid of parent zone			0.6
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
hypervisorType	see *cluster hypervisor type*			0.6
type	reserved field, don't evaluate it	true		0.6

Delete Cluster

Admins can use DeleteCluster to delete a cluster. For example:

```
DeleteCluster uuid=c1bd173d5cd84f0e9e7c47195ae27ec6
```

Danger: Deleting a cluster will delete hosts in the cluster; VMs will be migrated to other clusters or be stopped if no available clusters to migrate; primary storage and L2 networks attached to the cluster will be detached. There is no way to recover a deleted cluster.

Parameters

Name	Description	Optional	Choices	Since
uuid	cluster uuid			0.6
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6

Change State

Admins can use ChangeClusterState to change the state of a cluster. For example:

```
ChangeClusterState uuid=c1bd173d5cd84f0e9e7c47195ae27ec6 stateEvent=disable
```

Parameters

Name	Description	Optional	Choices	Since
uuid	cluster uuid			0.6
stateEvent	state trigger event • enable: change state to Enabled • disable: change state to Disabled		• enable • disable	0.6

Attach Primary Storage

Admins can use AttachPrimaryStorageToCluster command to attach a primary storage to a cluster. For example:

```
AttachPrimaryStorageToCluster clusterUuid=c1bd173d5cd84f0e9e7c47195ae27ec6 primaryStorageUuid=1b830f5
```

Note: Only sibling primary storage can be attached to a cluster. In other words, primary storage and clusters must be in the same zone.

Parameters

Name	Description	Optional	Choices	Since
clusterUuid	cluster uuid			0.6
primaryStorageUuid	primary storage uuid			0.6

Detach Primary Storage

Admin cans use DetachPrimaryStorageFromCluster to detach a primary storage from a cluster. For example:

```
DetachPrimaryStorageFromCluster clusterUuid=c1bd173d5cd84f0e9e7c47195ae27ec6 primaryStorageUuid=1b830
```

Note: During detaching, VMs that have root volumes on the primary storage and that run in the cluster will be stopped. Users can start those VMs again if the primary storage is still attached to some other clusters, or start them after the primary storage is attached to a new cluster.

Detaching primary storage is useful when admin wants to make a primary storage on longer accessible to a cluster. For example, in order to move VMs from a cluster equipped with aged hosts to a cluster with new, powerful hosts, admins can detach the primary storage on which root volumes of VMs locate from the old cluster and attach it to the new cluster, then start those stopped VMs; because the old cluster cannot access the primary storage anymore, ZStack will choose the new cluster to start VMs.

Parameters

Name	Description	Optional	Choices	Since
clusterUuid	cluster uuid			0.6
primaryStorageUuid	primary storage uuid			0.6

Attach L2 Network

Admin can use AttachL2NetworkToCluster command to attach a L2 network to a cluster. For example:

```
AttachL2NetworkToCluster clusterUuid=c1bd173d5cd84f0e9e7c47195ae27ec6 l2NetworkUuid=1b830f5bd1cb469b8
```

Note: Only sibling L2 networks can be attached to a cluster. In other words, L2 networks and clusters must be in the same zone.

Parameters

Name	Description	Optional	Choices	Since
clusterUuid	cluster uuid			0.6
l2NetworkUuid	L2 network uuid			0.6

Detach L2 Network

Admins can use DetachL2NetworkFromCluster command to detach a L2 network from a cluster. For example:

```
DetachL2NetworkFromCluster clusterUuid=c1bd173d5cd84f0e9e7c47195ae27ec6 l2NetworkUuid=1b830f5bd1cb469
```

Note: During detaching, VMs which run in the clusters and have nics on the L2 networks(through L3 networks) will be stopped. Users can start those VMs again if the L2 networks are still attached to other clusters, or start them after the L2 networks are attached to new clusters.

Detaching L2 networks can be useful when admins want to make network typology changes in datacenters. After hosts in a cluster no longer connect to a physical layer2 network, admin can detach the L2 network representing the physical layer2 network from the cluster.

Parameters

Name	Description	Optional	Choices	Since
clusterUuid	cluster uuid			0.6
l2NetworkUuid	L2 network uuid			0.6

Query Cluster

Admins can use QueryCluster to query clusters. For example:

```
QueryCluster hypervisorType=KVM
```

```
QueryCluster primaryStorage.availableCapacity>100000000
```

Primitive Fields of Query

see *cluster inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
zone	see *zone inventory*	parent zone	0.6
host	see *host inventory*	hosts belonging to this cluster	0.6
vmInstance	see *vm inventory*	VMs belonging to this cluster	0.6
l2Network	see *L2 network inventory*	L2 networks attached to this cluster	0.6
primaryStorage	see *primary storage inventory*	primary storage attached to this cluster	0.6

2.8.4 Tags

Admins can create user tags on a cluster with resourceType=ClusterVO. For example:

```
CreateUserTag resourceType=ClusterVO resourceUuid=80a979b9e0234564a22a4cca8c1dff43 tag=secureCluster
```

System Tags

Reserved Capacity

Tag	Description	Example	Since
host::reservedMemory::{capacity}	see *Host Capacity Reservation*	host::reservedMemory::1G	0.6

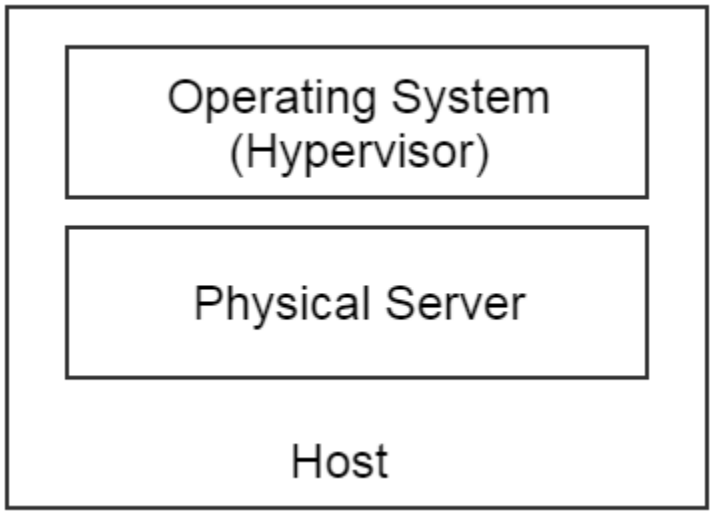
2.9 Host

Table of contents

- Host
  - Overview
  - Inventory
    - Properties
    - Example
    - Management IP
      - Management Network
    - State
      - Maintenance Mode
    - Status
    - State and Status
  - Operations
    - Add Host
      - Add KVM Host
      - Parameters
      - KVM Credentials
    - Delete Host
      - Parameters
    - Change Host State
      - Parameters
    - Reconnect Host
      - Parameters
    - Query Host
      - Primitive Fields of Query
      - Nested And Expanded Fields of Query
  - Global Configurations
    - load.all
    - load.parallelismDegree
    - ping.timeout
    - ping.parallelismDegree
    - connection.autoReconnectOnError
    - maintenanceMode.ignoreError
    - reservedCapacity.zoneLevel
    - reservedCapacity.clusterLevel
    - reservedCapacity.hostLevel
    - vm.migrationQuantity
    - reservedMemory
    - dataVolume.maxNum
    - host.syncLevel
  - Tags
    - System Tags
      - Host Capacity Reservation
      - Host Meta Data Information
      - KVM Host Meta Data Information

2.9.1 Overview

A host is a physical server installed with an operating system(hypervisor).



In ZStack, a host is the smallest unit providing computing resources that run VMs. Zones and clusters, which usually contain grouped hosts, are bigger units. Unlike its parent and ancestor both of which are logical resources, a host is a physical resource; many operations, which are seemingly applied to zones or clusters, are actually delegated to hosts. For example, when attaching a primary storage to a cluster, the real action performed might be mounting the primary storage on every host in the cluster.

Note: In this ZStack version, KVM is the only supported host

2.9.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
zoneUuid	uuid of ancestor zone. see *zone*			0.6
clusterUuid	uuid of parent cluster. see *cluster*			0.6
managementIp	see *management ip*			0.6
hypervisorType	see *cluster hypervisor type*			0.6
state	see *state*		• Enabled • Disabled • PreMaintenance • Maintenance	0.6
status	see *status*		• Connecting • Connected • Disconnected	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example


```
{
  "inventory": {
    "zoneUuid": "2893ce85c43d4a3a8d78f414da39966e",
    "name": "host1-192.168.0.203",
    "uuid": "43673938584447b2a29ab3d53f9d88d3",
    "clusterUuid": "8524072a4274403892bcc5b1972c2576",
    "description": "Test",
    "managementIp": "192.168.0.203",
    "hypervisorType": "KVM",
    "state": "Enabled",
    "status": "Connected",
    "createDate": "Jun 1, 2015 6:49:24 PM",
    "lastOpDate": "Jun 1, 2015 6:49:24 PM"
  }
}
```

Management IP

The management IP is used by ZStack management nodes to reach the operating systems(hypervisor) of hosts; depending on hypervisor types, it's necessary or not. For example, in VMWare, the official way to reach an ESXi host is through the VCenter Server, then the management IP is not necessary; however, in KVM, ZStack will deploy an agent to the Linux operating system, then the management IP is necessary.

Note: A management IP can be either an IP address or a DNS name, as long as the DNS name can be resolved by the operating systems on which ZStack management nodes run.

Note: In this ZStack version, as KVM is the only supported host, the management ip is a mandatory field.

Management Network

Though it's not enforced, it is recommended to have one or more dedicated subnets used as management networks. The Linux servers that run ZStack management nodes must be able to reach management networks, because management nodes need to send commands to hosts and other appliances on the management networks. In future chapters, we will see management network again when talking about appliance VMs, which are specific to *virtual router* in this ZStack version.

Warning: Specific to KVM, it's recommended to make all management IPs of hosts in the same zone be interreachable. In this ZStack version, there are no dedicated networks for VM migration; ZStack essentially uses management IPs to transfer data amid hosts during VM migrations. If hosts can not reach each other by management IPs, even they can be reached by ZStack management nodes, VM migrations among them will still fail.

State

Hosts have four states:

- **Enabled**:

  the state that allows VMs to be created, started, or migrated to

- **Disabled**:

  the state that DOESN'T allow VMs to be created, started, or migrated to

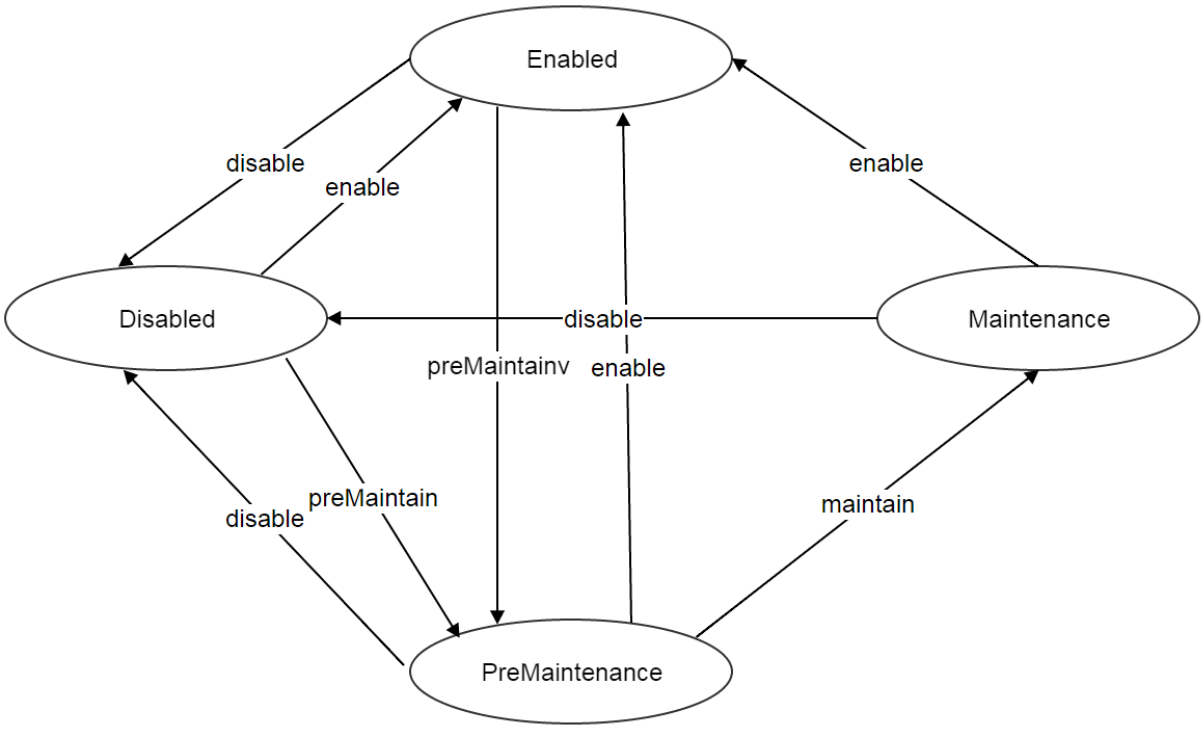
- **PreMaintenance**:

  the intermediate state indicating host is entering Maintenance state. See *maintenance mode*.

- **Maintenance**:

  the state indicating host has been in maintenance mode.

A state transition diagram is like:



Maintenance Mode

A host can be placed in maintenance mode when admins need to carry out maintenance work, for example, to install more memory. When a host is in the maintenance mode, neither API operations nor ZStack internal tasks can be performed to it. That is to say, tasks like starting VMs(API), stopping VMs(API), mounting primary storage(internal) cannot be performed. ZStack defines maintenance mode in two states: PreMaintenance and Maintenance. The sequence a host enters maintenance mode is shown as follows:

1. Changing the host's state to PreMaintenance. At this phase, ZStack will try to migrate all VMs running on the host to other appropriate hosts. If migrations fail, ZStack will stop those VMs.
2. After VMs are properly migrated or stopped, ZStack will change the host's state to Maintenance. Since now, admins can do maintenance work to the host.

Admins can take a host out of maintenance mode by placing it in Enabled or Disabled state, after maintenance work is done.

Note: When a host is in maintenance mode, admins can still attach primary storage or L2 networks to its parent cluster. Once the host quits maintenance mode, ZStack will send a reconnect message which will instruct the host to catch up work missed during it was in the maintenance mode; for example, mounting a NFS primary storage.

Status

A host's status reflects the status of command channel between the host and a ZStack management node. Command channels are the ways that ZStack management nodes communicate with hosts to perform operations. For example, in KVM, command channels are the HTTP connections between ZStack management nodes and Python agents running on hosts; in VMWare, command channels are connections between the VCenter Server and ESXi hosts.

Hosts have three status:

- **Connecting**:

  A ZStack management node is trying to establish the command channel between itself and the host. No operations can be performed to the host.

- **Connected**

  The Command channel has been successfully established between a ZStack management node and the host. Operations can be performed to the host. This is the only status that a host can start or create VMs.

- **Disconnected**

  The Command channel has lost between a ZStack management node and the host. No operations can be performed to the host.

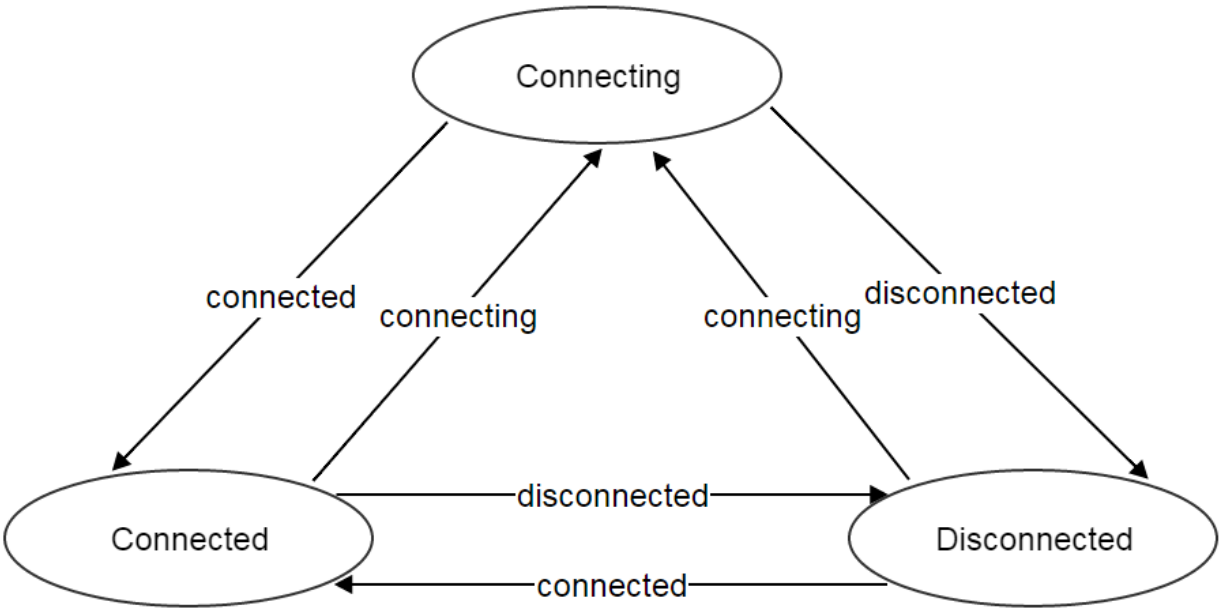
When booting, a ZStack management node will start the process of establishing the command channel to hosts it manages; in this stage, hosts's status are Connecting; after command channels are established, hosts' status change to Connected; if the management node fails to setup a command channel, or the command channel is detected as lost later on, the status of the host to which the command channel connect changes to Disconnected.

ZStack management nodes will periodically send ping commands to hosts to check health of command channels; once a host fails to respond, or a ping command times out, the host's status changes to Disconnected.

Note: ZStack will keep sending ping commands to a disconnected host. Once the host recovers and responds to the ping command, ZStack will reestablish the command channel and alter the host to Connected. So when a host is physically removed from a cloud, please remember to delete it from ZStack, otherwise ZStack management nodes will keep pinging it.

Note: No ping command will be sent if a host is in maintenance mode.

A status transition diagram is like:



State and Status

There are no direct relations between states and status. States represent admin's decisions to a host, while status represents communication condition of a host.

2.9.3 Operations

Add Host

The commands adding a host varies for different hypervisors.

Add KVM Host

Admins can use AddKVMHost to add a KVM host. For example:

```
AddKVMHost clusterUuid=8524072a4274403892bcc5b1972c2576 managementIp=192.168.10.10 name=kvm1 username
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
clusterUuid	uuid of parent cluster, see *cluster*			0.6
managementIp	see *management ip*			0.6
username	see *kvm credentials*			0.6
password	see *kvm credentials*			0.6

KVM Credentials ZStack uses a Python agent called kvmagent to manage KVM hosts. To make things full automation, ZStack utilizes Ansible to configure target Linux operating systems and deploy kvmagents; and to bootstrap Ansible on target Linux operating systems, ZStack needs SSH username/password of **root** user to inject SSH public keys in KVM hosts in order to make Ansible work without prompting username/password. The **root** privilege is required as both Ansible and kvmagent need full control of the system.

Delete Host

Admins can use DeleteHost command to delete a host. For example:

```
DeleteHost uuid=2893ce85c43d4a3a8d78f414da39966e
```

> **Danger:** Deleting hosts will stop all VMs on the host. There is no way to recover a deleted host.

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	host uuid			0.6

Change Host State

Admins can use ChangeHostState command to change a host's state. For example:

```
ChangeHostState stateEvent=preMaintain uuid=2893ce85c43d4a3a8d78f414da39966e
```

Parameters

Name	Description	Optional	Choices	Since
uuid	host uuid			0.6
stateEvent	state trigger event. See *state* **Note:** The state trigger event 'maintain' shown in *state* section is used internally and is not available in the API.		• enable • disable • preMaintain	0.6

Reconnect Host

Admins can use ReconnectHost to re-establish the command channel between a ZStack management node and a host. For example:

```
ReconnectHost uuid=2893ce85c43d4a3a8d78f414da39966e
```

See *status* for details.

Parameters

Name	Description	Optional	Choices	Since
uuid	host uuid			0.6

Query Host

Admins can use QueryHost to query hosts. For example:

```
QueryHost managementIp=192.168.0.100
```

```
QueryHost vmInstance.vmNics.ip=10.21.100.2
```

Primitive Fields of Query

see *host inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
zone	*zone inventory*	ancestor zone	0.6
cluster	*cluster inventory*	parent cluster	0.6
vmInstance	*VM inventory*	VMs running on this host	0.6

2.9.4 Global Configurations

load.all

Name	Category	Default Value	Choices
load.all	host	true	• true • false

Whether to connect all hosts when management nodes boot. If set to true, management nodes will connect to all hosts simultaneously during booting time, which may exhaust resources of the machines running management nodes if there are a large number of hosts in the cloud; if set to false, accompanying with *load.parallelismDegree*, management nodes will connect a portion of hosts each time and repeat until all hosts are connected.

load.parallelismDegree

Name	Category	Default Value	Choices
load.parallelismDegree	host	100	> 0

When *load.all* is set to false, this configuration defines the number of hosts that management nodes will connect simultaneously during booting time.

ping.timeout

Name	Category	Default Value	Choices
ping.interval	host	60	> 0

The interval that management nodes periodically send ping commands to hosts in order to check connection status, in seconds.

ping.parallelismDegree

Name	Category	Default Value	Choices
ping.parallelismDegree	host	100	> 0

The parallel degree that management nodes send ping commands. If the amount of hosts are larger than this value, management nodes will repeat until all hosts are pinged. For example, ping first 100 hosts, then ping second 100 hosts ...

connection.autoReconnectOnError

Name	Category	Default Value	Choices
connection.autoReconnectOnError	host	true	• true • false

Whether to reconnect hosts when their status change from Connected to Disconnected. If set to true, management nodes will reconnect hosts whose status change from Connected to Disconnected by ping commands, in order to catch up with operations missed during hosts in disconnected; if set to false, management nodes will not automatically reconnect them, admins may need to manually do it if necessary.

maintenanceMode.ignoreError

Name	Category	Default Value	Choices
maintenanceMode.ignoreError	host	false	• true • false

Whether to ignore errors happening during hosts enter maintenance mode. If set to true, errors are ignored and hosts always successfully enter maintenance mode; if set to false, hosts will fail to enter maintenance mode if any error happens, for example, failing to migrate a VM.

reservedCapacity.zoneLevel

Name	Category	Default Value	Choices
reservedCapacity.zoneLevel	hostAllocator	true	• true • false

Whether to enable host capacity reservation at zone level; see *host capacity reservation*.

reservedCapacity.clusterLevel

Name	Category	Default Value	Choices
reservedCapacity.clusterLevel	hostAllocator	true	• true • false

Whether to enable host capacity reservation at cluster level; see *host capacity reservation*.

reservedCapacity.hostLevel

Name	Category	Default Value	Choices
reservedCapacity.hostLevel	hostAllocator	true	• true • false

Whether to enable host capacity reservation at host level; see *host capacity reservation*.

vm.migrationQuantity

Name	Category	Default Value	Choices
vm.migrationQuantity	kvm	2	> 0

The number that how many VMs can be migrated in parallel when KVM hosts enter maintenance mode.

reservedMemory

Name	Category	Default Value	Choices
reservedMemory	kvm	512M	>= 0

A string that memory capacity reserved on KVM hosts if *reservedCapacity.hostLevel* is set to true. The value is a number followed by a unit character that can be one of B/K/M/G/T; if no unit character followed, the number is treated as bytes.

dataVolume.maxNum

Name	Category	Default Value	Choices
dataVolume.maxNum	kvm	24	0 - 24

The max number of data volumes that can be attached to VMs of hypervisor type – KVM.

host.syncLevel

Name	Category	Default Value	Choices
host.syncLevel	kvm	10	> 2

The max number of concurrent commands that can be simultaneously executed on KVM hosts.

2.9.5 Tags

Admins can create user tags on a host with resourceType=HostVO. For example:

```
CreateUserTag tag=largeMemoryHost resourceUuid=0a9f95a659444848846b5118e15bff32 resourceType=HostVO
```

System Tags

Host Capacity Reservation

Admins can use system tags to reserve a portion of memory on hosts for system software. ZStack provides various system tags and global configurations for fine-grained memory reservation policies:

- **Hypervisor Global Level**:

  The global configuration *reservedMemory* applies to all KVM hosts if not overridden by settings of other levels.

- **Zone Level**:

  See *zone host::reservedMemory*; the value of this system tag applies to all hosts in the zone if not overridden by settings of other levels. This overrides global level.

- **Cluster Level**:

  See *cluster host::reservedMemory*; the value of this system tag applies to all hosts in the cluster if not overridden by the setting of host level. This overrides zone level and global level.

- **Host Level**:

Tag	Description	Example	Since
reservedMemory::{capacity}	reserved memory on this host.	reservedMemory::1G	0.6

  this overrides all above levels.

For example, assuming you have 3 KVM hosts in zone1->cluster1->{host1, host2, host3}; by default the memory reservation is controlled by the global configuration *reservedMemory* that defaults to 512M; then you create a system tag *host::reservedMemory::1G* on zone1, so memory reservation on all 3 hosts is 1G now; then you create a system tag *host::reservedMemory::2G* on cluster1, memory reservation of 3 hosts changes to 2G; finally, you create a system tag *reservedMemory::3G* on host1, then memory reservation is 3G on host1 but still 2G on host2 and host3.

Host Meta Data Information

Tag	Description	Example	Since
capability:liveSnapshot	if present, the host's hypervisor supports live volume snapshot	capability:liveSnapshot	0.6
os::distribution::{distribution}	OS distribution of the host	os::distribution::Ubuntu	0.6
os::release::{release}	OS release of the host	os::release::trusty	0.6
os::version::{version}	OS version the host	os::version::14.04	0.6

KVM Host Meta Data Information

Tag	Description	Example	Since
qemu-img::version::{version}	qemu-img version	qemu-img::version::2.0.0	0.6
libvirt::version::{version}	libvirt version	libvirt::version::1.2.2	0.6
hvm::{flag}	host hardware virtualization flag; vmx means Intel CPU; svm means AMD CPU	hvm::vmx	0.6

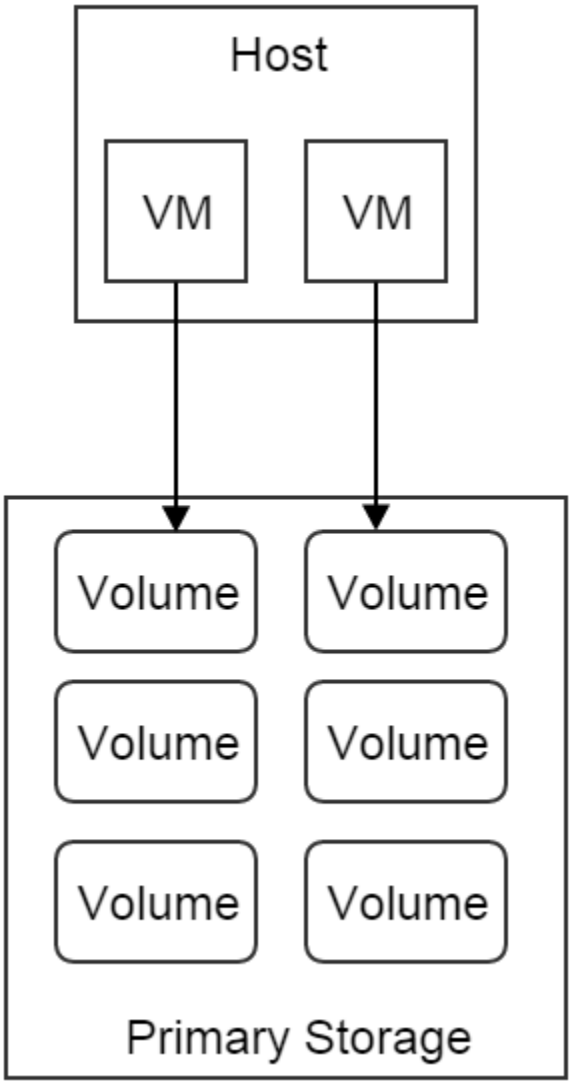
2.10 Primary Storage

Table of contents

- *Primary Storage*
  - *Overview*
  - *Inventory*
    - *Properties*
    - *Example*
      - *Capacity*
      - *NFS Capacity*
      - *URL*
      - *NFS URL*
    - *State*
    - *Status*
    - *State and Status*
    - *Attaching Cluster*
  - *Operations*
    - *Add Primary Storage*
      - *Add NFS Primary Storage*
      - *Properties*
    - *Delete Primary Storage*
      - *Properties*
    - *Change Primary Storage State*
      - *Properties*
    - *Attach Cluster*
    - *Detach Cluster*
    - *Query Primary Storage*
      - *Primitive Fields of Query*
      - *Nested And Expanded Fields of Query*
  - *Global Configurations*
    - *mount.base*
  - *Tags*
    - *System Tags*
      - *Storage Volume Snapshot*

2.10.1 Overview

A primary storage is the storage system in datacenter that stores disk volumes for VMs. Primary storage can be local disks(e.g. hard drives of hosts) or network shared (e.g. NAS, SAN) storage.



A primary storage stores volumes for VMs running in clusters that have been attached to this primary storage.

A primary storage can be attached to only sibling clusters.

Note: In this ZStack version, NFS is the only supported primary storage

2.10.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
zoneUuid	uuid of parent zone, see *zone*			0.6
totalCapacity	total disk capacity, in bytes, see *capacity*			0.6
availableCapacity	available disk capacity, in bytes, see *capacity*			0.6
url	see *url*			0.6
type	primary storage type		• NFS	0.6
state	see *state*		• Enabled • Disabled	0.6
status	see *status*		• Connecting • Connected • Disconnected	0.6
attachedClusterUuids	a list of cluster uuid to which the primary storage has been attached, see *attach cluster*			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example

```
{
  "inventory": {
    "uuid": "f4ac0a3119c94c6fae844c2298615d27",
    "zoneUuid": "f04caf351c014aa890126fc78193d063",
    "name": "nfs",
    "url": "192.168.0.220:/storage/nfs",
    "description": "Test Primary Storage",
    "totalCapacity": 10995116277768819,
    "availableCapacity": 10995162768,
    "type": "NFS",
    "state": "Enabled",
```

```
    "mountPath": "/opt/zstack/f4ac0a3119c94c6fae844c2298615d27",
    "createDate": "Jun 1, 2015 2:42:51 PM",
    "lastOpDate": "Jun 1, 2015 2:42:51 PM",
    "attachedClusterUuids": [
      "f23e402bc53b4b5abae87273b6004016",
      "4a1789235a86409a9a6db83f97bc582f",
      "fe755538d4e845d5b82073e4f80cb90b",
      "1f45d6d6c02b43bfb6196dcacb5b8a25"
    ]
  }
}
```

Capacity

ZStack keeps tracking disk capacities of primary storage in order to select suitable one to create volumes. The capacities reported by different primary storage plugins may be different; for example, for those supporting over-provisioning, the capacity reported may be larger than real; for those not supporting over-provisioning, the capacity reported may be equal to or smaller than real.

NFS Capacity NFS doesn't support over-provisioning, so the capacity is counted by volumes' virtual sizes using below formulas:

```
totalCapacity = NFS's total capacity
availableCapacity = totalCapacity - sum(volumes' virtual sizes)
```

Volumes' virtual sizes will be discussed in chapter *volume*; for those impatient, a volume's virtual size is the size when a volume is fully filled; for example, when you created a volume with 1G capacity, before it's fully filled, its real size may be 10M because of thin-provisioning technology.

URL

A URL is a string that contains information needed by primary storage plugins for manipulating storage systems. Although it's named as URL, the certain format of the string is up to primary storage types and is not necessary to strictly follow the URL convention, to give flexibilities to plugins to encode information that may not be able to fit in the URL format.

NFS URL For NFS primary storage, the URL is encoded as:

```
ip-or-dns-name-of-nfs-server:/absolute-path-to-directory
```

For example:

```
192.168.0.220:/storage/nfs/
```

State

Primary storage has two states:

- **Enabled**:

  the state that allows volumes to be created

- **Disabled**:

  the state that DOESN'T allow volumes to be created

Status

Like *host status*, primary storage status reflect the status of command channels amid ZStack management nodes and primary storage. Command channels are the ways ZStack management nodes communicate with storage systems that primary storage represent; depending on primary storage types, for example, it can be HTTP connections among ZStack management nodes and primary storage agents or communication methods provided by storage SDKs.

There are three status:

- **Connecting**:

  A ZStack management node is trying to establish the command channel between itself and the primary storage. No operations can be performed to the primary storage.

- **Connected**

  The command channel has been successfully established between a ZStack management node and the primary storage. Operations can be performed to the primary storage.

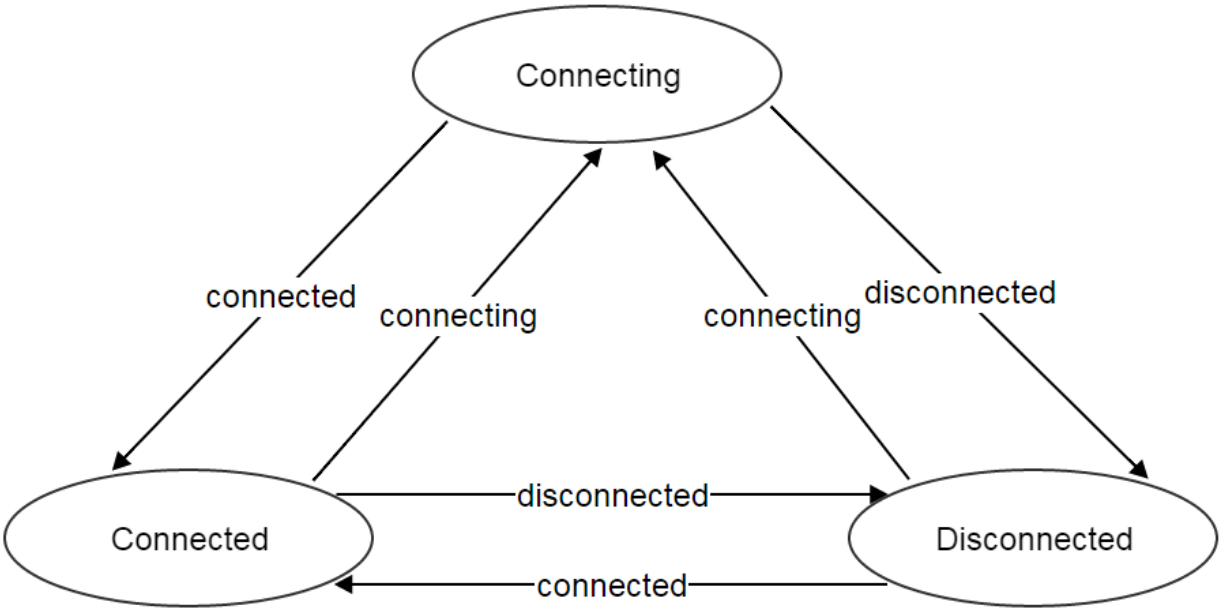
- **Disconnected**

  The command channel has lost between a ZStack management node and the primary storage. No operations can be performed to the primary storage.

ZStack management nodes will try to establish command channels when booting and will periodically send ping commands to primary storage to check health of command channels during running; once a primary storage fails to respond, or a ping command times out, the command channel is considered as lost and the primary storage will be placed in Disconnected.

Note: ZStack will keep sending ping commands when a primary storage is in status of Disconnected. Once the primary storage recovers and responds to ping commands, ZStack will reestablish the command channel and place the primary storage in status of Connected. So when a primary storage is physically removed from the cloud, please delete it from ZStack, otherwise ZStack will keep pinging it.

Here is the transition diagram:

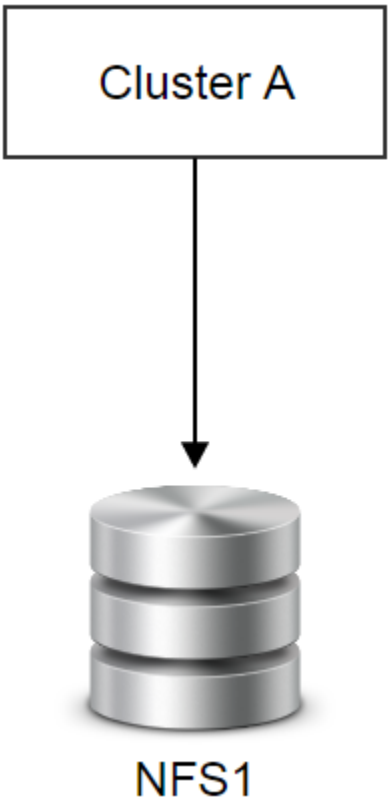


State and Status

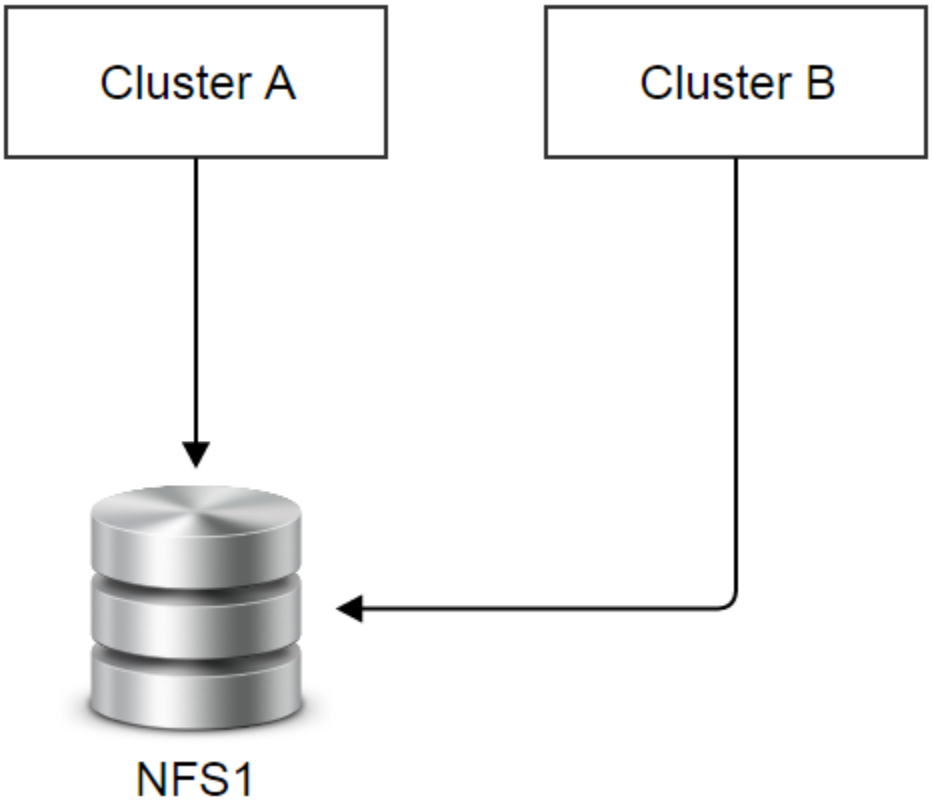
There is no direct relations between states and status. States represent admin's decisions to primary storage, while status represent communication conditions of primary storage.

Attaching Cluster

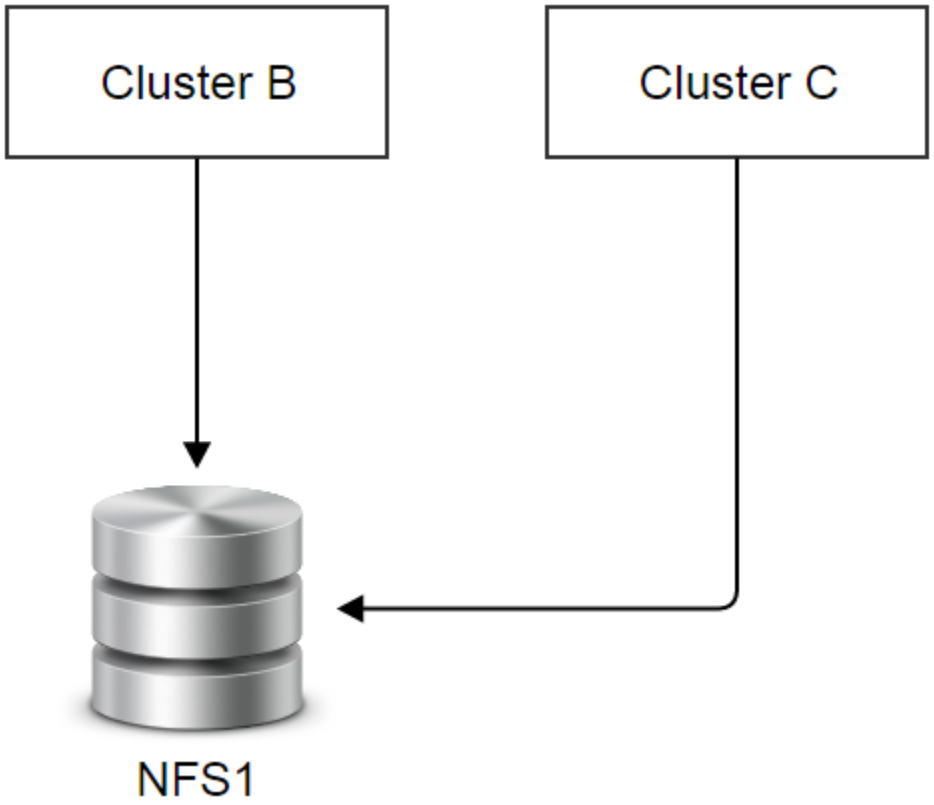
Attaching clusters is to associate primary storage to sibling clusters, which provides a flexible way that manifests relations between hosts and storage systems in a real datacenter. Let's see a concreted example; assuming you have a cluster (cluster A) attached to a NFS primary storage (NFS1), like below diagram:



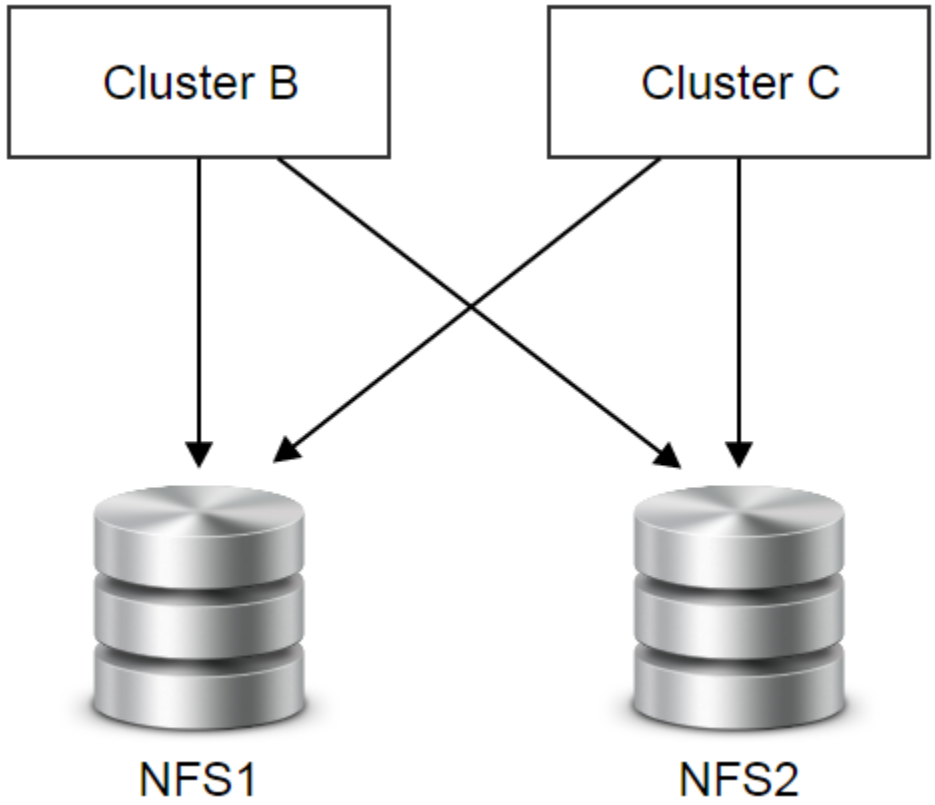
Some time later, the cluster A is running out of memory but the primary storage still have plenty of disk spaces, so you decide to add another cluster (cluster B) which will also use NFS1; then you can create cluster B and attach NFS1 to it.



After running a while, the hardware of cluster A is getting outdated and you decide to retire them; you add a new powerful cluster (cluster C) attached to NFS1 and place all hosts in cluster A into maintenance mode, so all VMs running in cluster A are migrated to cluster B or cluster C; lastly, you detach NFS1 from cluster A and delete it. Now the datacenter looks like:



Finally, NFS1 starts running out of capacity, you add one more primary storage (NFS2), and attach it to both cluster B and cluster C.



2.10.3 Operations

Add Primary Storage

The commands adding a primary storage varies for different types of primary storage.

Add NFS Primary Storage

Admins can use AddNfsPrimaryStorage to add a NFS primary storage. For example:

```
AddNfsPrimaryStorage name=nfs1 zoneUuid=1b830f5bd1cb469b821b4b77babfdd6f url=192.168.0.220:/storage/n
```

Properties

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
zoneUuid	uuid of parent zone, see *zone*			0.6
url	see *url*			0.6

Delete Primary Storage

Admins can use DeletePrimaryStorage to delete a primary storage. For example:

```
DeletePrimaryStorage uuid=2c830f5bd1cb469b821b4b77babfdd6f
```

Danger: Deleting a primary storage will delete all volumes and volume snapshots it contains. VMs will be deleted as results of deleting root volumes. There is no way to recover a deleted primary storage. Clusters attached will be detached.

Properties

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	primary storage uuid			0.6

Change Primary Storage State

Admins can use ChangePrimaryStorageState to change the state of a primary storage. For example:

```
ChangePrimaryStorageState stateEvent=enable uuid=2c830f5bd1cb469b821b4b77babfdd6f
```

Properties

Name	Description	Optional	Choices	Since
uuid	primary storage uuid			0.6
stateEvent	state trigger event • enable: change state to Enabled • disable: change state to Disabled		• enable • disable	0.6

Attach Cluster

See *Attach Primary Storage*.

Detach Cluster

See *Detach Primary Storage*.

Query Primary Storage

Admins can use QueryPrimaryStorage to query primary storage. For example:

```
QueryPrimaryStorage totalCapacity<100000000000
```

```
QueryPrimaryStorage volumeSnapshot.uuid?=13238c8e0591444e9160df4d3636be82,33107835aee84c449ac04c96228
```

Primitive Fields of Query

see *primary storage inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
zone	*zone inventory*	parent zone	0.6
volume	*volume inventory*	volumes on this primary storage	0.6
volumeSnapshot	*volume snapshot inventory*	volume snapshots on this primary storage	0.6
cluster	*cluster inventory*	clusters the primary storage is attached to	0.6

2.10.4 Global Configurations

mount.base

Name	Category	Default Value	Choices
mount.base	nfsPrimaryStorage	/opt/zstack/nfsprimarystorage	absolute path that starts with '/'

The mount point that NFS primary storage is mounted on the KVM hosts.

Note: Changing this value only affect new NFS primary storage

2.10.5 Tags

Users can create user tags on a primary storage with resourceType=PrimaryStorageVO. For example:

```
CreateUserTag resourceType=PrimaryStorage tag=SSD resourceUuid=e084dc809fec4092ab0eff797d9529d5
```

System Tags

Storage Volume Snapshot

Tag	Description	Example	Since
capability:snapshot	if present, the primary storage supports storage volume snapshot	capability:snapshot	0.6

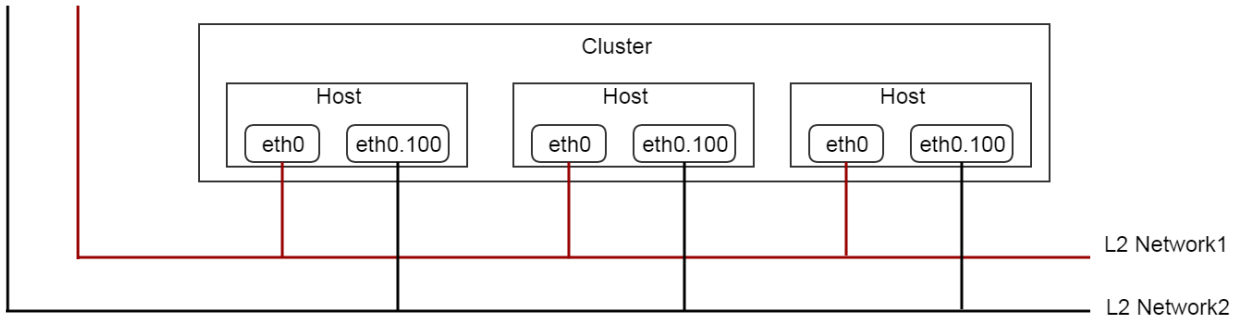
2.11 L2 Network

Table of contents

- *L2 Network*
  - *Overview*
  - *Inventory*
    - *Properties*
    - *Physical Interface*
    - *Attaching Cluster*
    - *L2NoVlanNetwork*
      - *L2NoVlanNetwork KVM Specific*
      - *L2NoVlanNetwork Inventory Example*
    - *L2VlanNetwork*
      - *L2VlanNetwork KVM Specific*
      - *L2VlanNetwork Inventory Example*
  - *Operations*
    - *Create L2 Network*
    - *Create L2NoVlanNetwork*
      - *Parameters*
    - *Delete L2 Network*
      - *Parameters*
    - *Attach Cluster*
    - *Detach Cluster*
    - *Query L2 Network*
      - *Primitive Fields of Query*
      - *Nested and Expanded Fields of Query*
  - *Tags*

2.11.1 Overview

A L2 network reflects a layer2 broadcast domain in a datacenter. That means, in addition to the traditional OSI data link layer, all technologies that provide layer 2 isolation can be L2 networks in ZStack. For example, VLAN, VxLAN, or SDNs that create layer 2 overlay networks. In ZStack, a L2 network is responsible for providing the layer 2 isolation method to child L3 networks.



A L2 network can be attached to sibling clusters.

2.11.2 Inventory

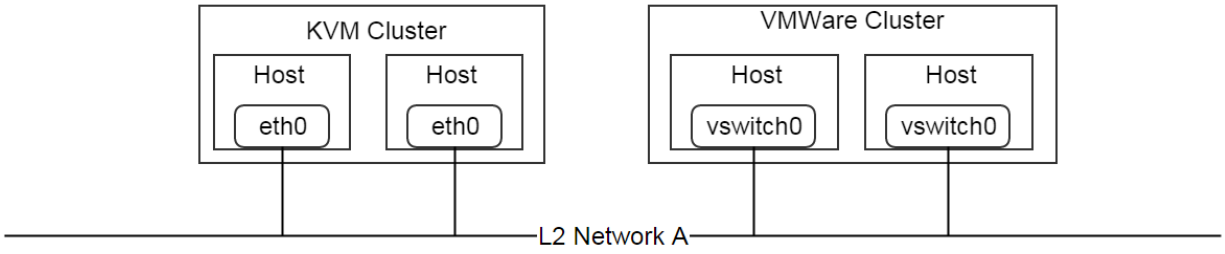
Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
zoneUuid	uuid of parent zone, see *zone*			0.6
physicalInterface	see *physical interface*			0.6
type	L2 network type		• L2NoVlanNetwork • L2VlanNetwork	0.6
attachedClusterUuids	a list of cluster uuid to which the L2 network has attached, see *attach cluster*			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Physical Interface

The physical interface is a string that contains information needed by a L2 network plugin for manipulating network system in a datacenter. The information encoded in physical interface is specific to L2 network types and hypervisor types of clusters that L2 networks may attach. This sounds a little complex. The complexity is originated from hypervisors using their own notations to describe L2 networks, and a L2 network can be attached to multiple clusters of different hypervisor types. A real world example may help to understand this.

Let's say your datacenter has a L2 network (l2Network A) which spans to two clusters, one is a KVM cluster, another is a VMWare cluster. In KVM, the L2 network is realized by ethernet device in Linux operating system; in this example, let's assume each eth0 of KVM hosts connects to the L2 network. In the VMWare cluster, the L2 network is realized by vswitch; in this example, let's assume vswitch0 in the VMWare cluster connects to the L2 network; then the typology is like:



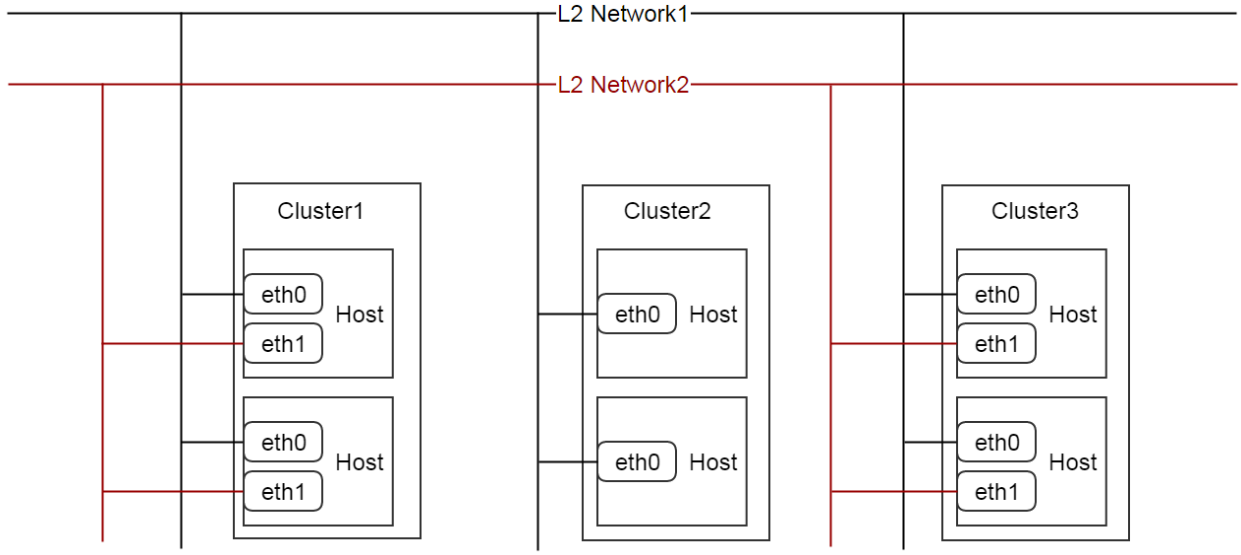
As mentioned in section *host*, lots of operations seemingly applied to clusters are actually delegated to hosts; Here, when attaching the L2 network A to the KVM cluster and the VMWare cluster, ZStack must understand what are

notations describing the L2 network in those hypervisors of clusters; in this case, ZStack must know that on KVM hosts, eth0 is the representation of the L2 network, but on VMWare hosts, vswitch0 is the representation. Physical interface is the field that encodes those hypervisor specific information.

Note: As this ZStack version supports only KVM, we won't discuss VMWare details for L2 networks. Above example largely aims to help understand the design of the physical interface.

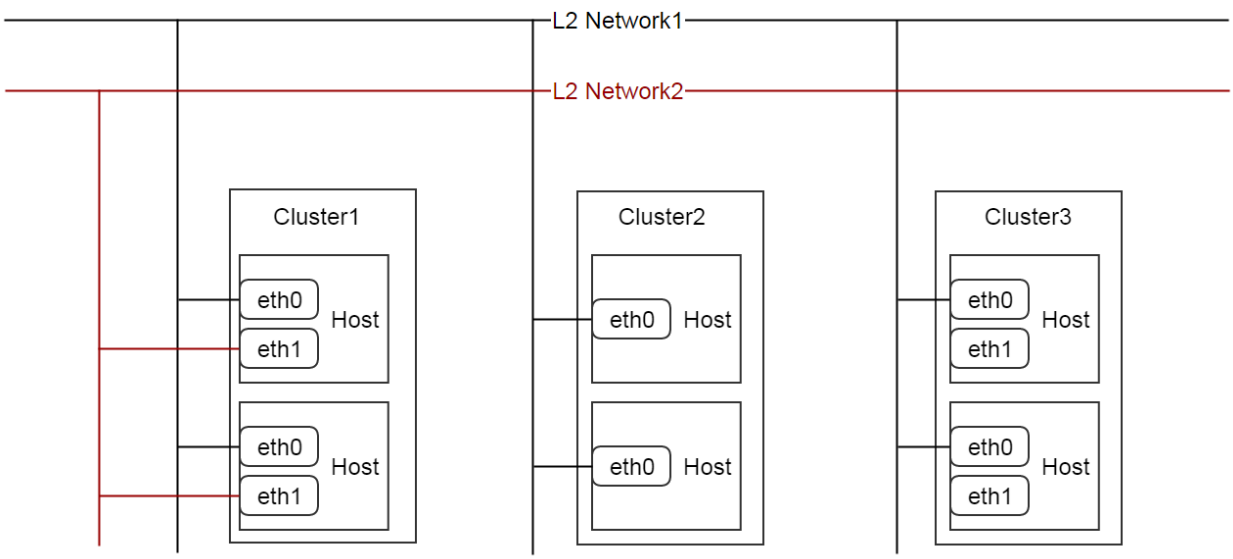
Attaching Cluster

Attaching cluster is to associate L2 networks to sibling clusters, which provides a flexible way that manifests relations between hosts and layer 2 networks in a real datacenter. Let's see a concrete example.



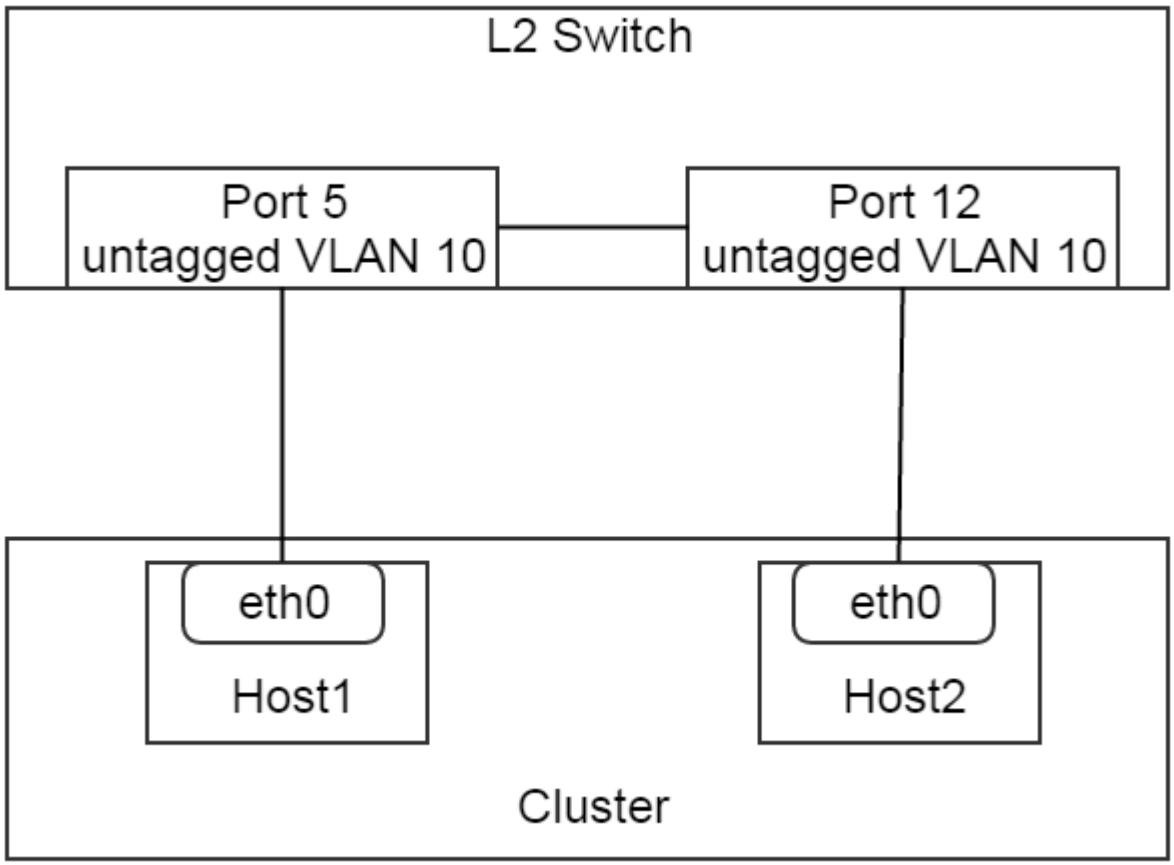
Let's assume the network typology in your datacenter is as above diagram. Eth0 of hosts in all clusters are on the same layer 2 network called L2 Network1; eth1 of cluster1 and cluster3 are on another layer 2 network called L2 network2. To describe this typology in ZStack, you can attach L2 network1 to all three clusters but attach L2 network2 to only cluster1 and cluster3.

A couple months later, the network typology needs changing because of business requirements, you unplug cables of eth1 of hosts in cluster3 from the rack switch, so cluster3 is not with L2 network2 anymore; you can detach the L2 network2 from cluster3 to notify ZStack about the network typology change.

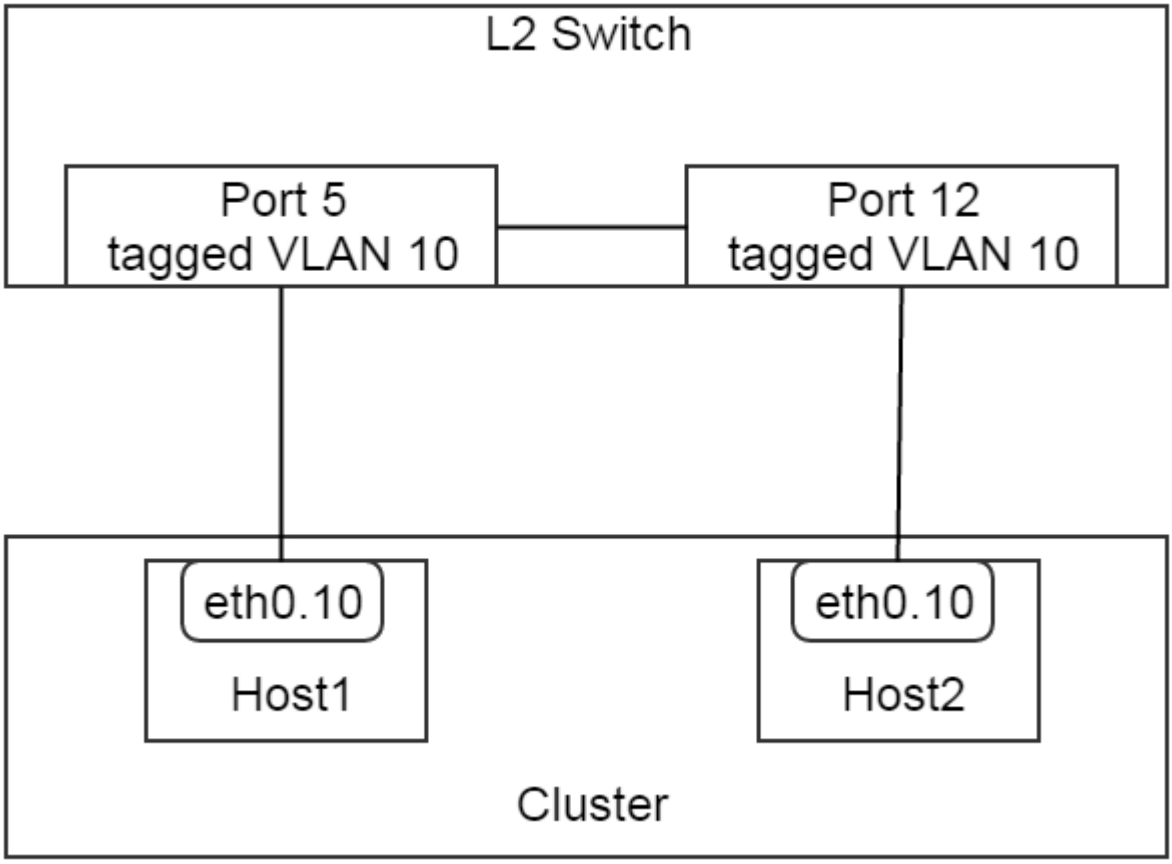


L2NoVlanNetwork

L2NoVlanNetwork, whose properties are listed in *properties* is the base type of L2 Networks. The 'NoVlan' in the name DOESN'T mean the network cannot use VLAN technology, it only denotes that ZStack itself will not use VLAN to create a layer 2 broadcast domain in an active manner. To make it clear, take a look at below two diagrams:



In this setup, two switch ports 5 and 12 are untagged with VLAN 10(access port with VLAN 10 in Cisco term), and connect to eth0 on host1 and host2 respectively. This is a very valid setup matching to a L2NoVlanNetwork. Admin cans create a L2NoVlanNetwork with 'physicalInterface' = 'eth0' and attach it to the cluster.



In this setup, two switch ports 5 and 12 are tagged with VLAN 10(trunk port with VLAN 10 in Cisco term), and respectively connect to eth0.10 that is a pre-created VLAN device on host1 and host2. This is also a very valid setup matching to a L2NoVlanNetwork. Admins can create a L2NoVlanNetwork with 'physicalInterface' = 'eth0.10' and attach it to the cluster.

Now it should be understood that a L2NoVlanNetwork maps to a pre-created layer 2 broadcast domain; ZStack won't create any new broadcast domain for L2NoVlanNetwork.

L2NoVlanNetwork KVM Specific

When attaching a L2NoVlanNetwork to a KVM cluster, the *physicalInterface* should be the ethernet device name in the Linux operating system; for example, eth0, eth0.10, em1. ZStack will use 'physicalInterface' as device name when creating a bridge using brctl. The pseudo operations are like:

```
Assuming physicalInterface = eth0

brctl create br_eth0
brctl addif br_eth0 eth0
```

Note: If you have multiple clusters of hosts whose ethernet devices connect to the same L2 network, and you want to attach that L2 network to those clusters, please make sure names of all ethernet devices are the same among all Linux

operating systems on hosts. For example, all ethernet devices are named as eth0. The best practice is installing the same Linux system on hosts of those clusters, or using udev to make all device names same.

L2NoVlanNetwork Inventory Example

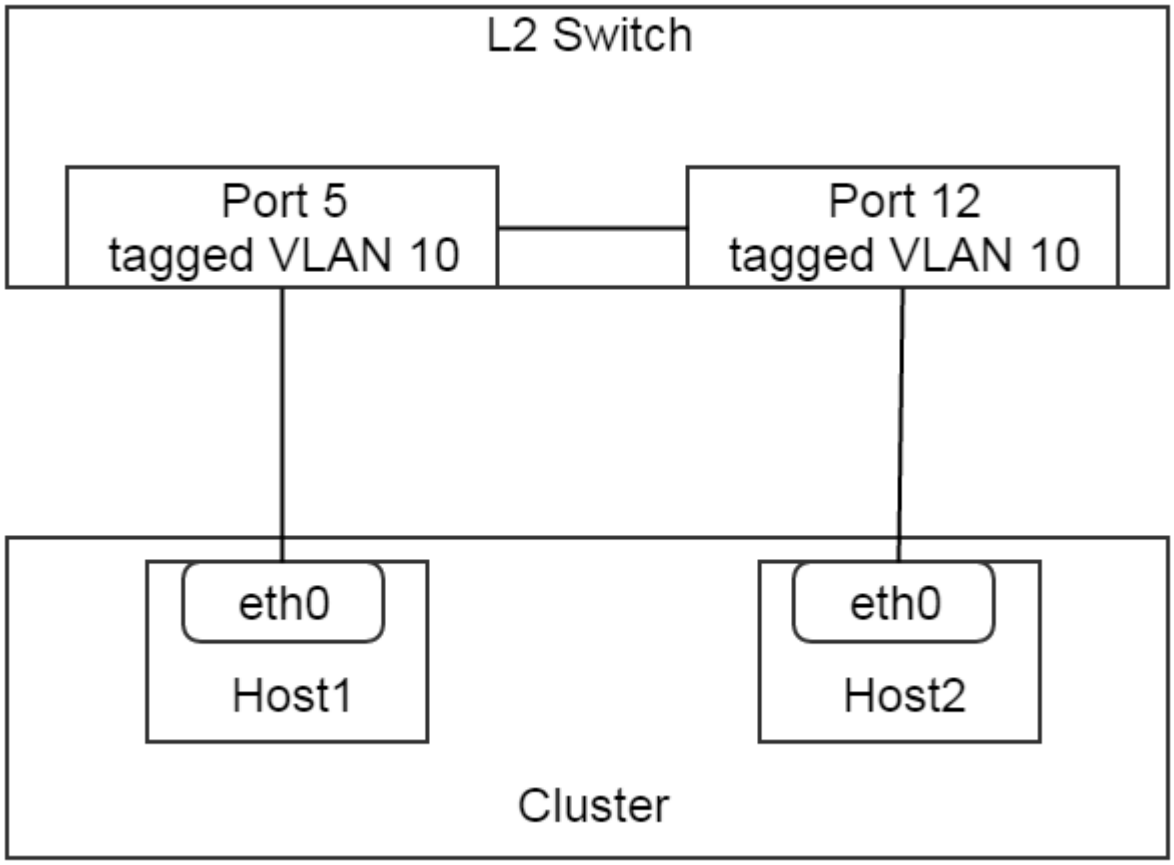
```
{
  "inventory": {
    "uuid": "f685ff94513542bbb8e814027f8deb13",
    "name": "l2-basic",
    "description": "Basic L2 Test",
    "zoneUuid": "45a2864b6ddf4d2fb9b4c3736a923dcb",
    "physicalInterface": "eth0",
    "type": "L2NoVlanNetwork",
    "createDate": "Jun 1, 2015 12:58:35 PM",
    "lastOpDate": "Jun 1, 2015 12:58:35 PM",
    "attachedClusterUuids": []
  }
}
```

L2VlanNetwork

A L2VlanNetwork is a L2 network that ZStack will actively use a VLAN to create a layer 2 broadcast domain. The ways that ZStack create layer 2 broadcast domains depend on hypervisor types of clusters, to which L2 networks are going to attach. In addition to *properties*, a L2VlanNetwork has one more property:

Name	Description	Optional	Choices	Since
vlan	VLAN id used to create layer 2 broadcast domain		[0, 4095]	0.6

When attaching a L2VlanNetwork to a cluster, ZStack uses 'vlan' collaborating with 'physicalInterface' to create vlan devices on hosts in the cluster; in order to make this work, the switch ports to which ethernet devices identified by 'physicalInterface' connect must be tagged with 'vlan'. For example:



In this setup, switch ports 5 and 12 have been tagged with VLAN 10, then admins can create a L2VlanNetwork with 'physicalInterface' = 'eth0' and 'vlan' = 10 and attach it to the cluster.

L2VlanNetwork KVM Specific

When attaching a L2VlanNetwork to a KVM cluster, ZStack will create VLAN devices on all hosts in the cluster then create bridges. The pseudo operations are like:

```
Assuming physicalInterface = eth0, vlan = 10

vconfig add eth0 10
brctl create br_eth0_10
brctl addif br_eth0_10 eth0.10
```

Note: Like L2NoVlanNetwork, please make sure ethernet device names of all hosts in clusters to which a L2VlanNetwork is about to attach are the same.

L2VlanNetwork Inventory Example

```
{
    "inventory": {
      "vlan": 10,
```

```
      "uuid": "14a01b0978684b2ea6e5a355c7c7fd73",
      "name": "TestL2VlanNetwork",
      "description": "Test",
      "zoneUuid": "c74f8ff8a4c5456b852713b82c034074",
      "physicalInterface": "eth0",
      "type": "L2VlanNetwork",
      "createDate": "Jun 1, 2015 4:31:47 PM",
      "lastOpDate": "Jun 1, 2015 4:31:47 PM",
      "attachedClusterUuids": []
    }
}
```


2.11.3 Operations

Create L2 Network

The commands creating L2 networks vary for different L2 network types.

Create L2NoVlanNetwork

Admins can use CreateL2NoVlanNetwork to create a L2NoVlanNetwork. For example:

```
CreateL2NoVlanNetwork name=management-network physicalInterface=eth0 zoneUuid=9a94e647a9f64bb392afcdc
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
zoneUuid	uuid of parent zone, see *zone*			0.6
physicalInterface	see *physical interface*			0.6

Delete L2 Network

Admins can use DeleteL2Network to delete a L2 network. For example:

```
DeleteL2Network uuid=a5535531eb7346ce89cfd7e643ad1ef8
```

Danger: Deleting a L2 network will cause its child L3 network to be deleted. For consequences of deleting L3 networks, see *Delete L3 Network*. There is no way to recover a deleted L2 network.

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	L2 network uuid			0.6

Attach Cluster

See *Attach L2 Network*.

Detach Cluster

See *Detach L2 Network*.

Query L2 Network

Admins can use QueryL2Network to query L2 networks. For example:

```
QueryL2Network physicalInterface=eth0
```

```
QueryL2Network l3Network.ipRanges.startIp=192.168.0.2
```

Primitive Fields of Query

see *L2 network inventory*.

Nested and Expanded Fields of Query

Field	Inventory	Description	Since
l3Network	*L3 network inventory*	L3 networks belonging to this L2 network	0.6
cluster	*cluster inventory*	clusters this L2 network is attached to	0.6
zone	*zone inventory*	parent zone	0.6

2.11.4 Tags

Admins can create user tags on a L2 network with resourceType=L2NetworkVO. For example:

```
CreateUserTag resourceType=L2NetworkVO tag=publicL2 resourceUuid=cff4be8694174b0fb831a9fe53b1d62b
```

2.12 L3 Network

Table of contents

- L3 Network
  - Overview
    - Subnet
    - Network Services
  - Inventory
    - Properties
    - Example
    - State
    - DNS Domain
    - IP Range
      - Inventory
      - Properties
      - Example
    - DNS
    - L2 Networks and L3 Networks
    - Network Service References
      - Inventory
      - Properties
      - Example
  - Network Typology
  - Operations
    - Create L3 Network
      - Parameters
    - Type
    - System L3 Network
    - Delete L3 Network
      - Parameters
    - Add IP Ranges
      - Add Split Ranges
      - Parameters
      - Add CIDR
      - Parameters
    - Delete IP Range
      - Parameters
    - Add DNS
      - Parameters
    - Attach Network Service
      - Parameters
    - Query L3 Network
      - Primitive Fields of Query
      - Nested And Expanded Fields of Query
  - L3 Network Tags
  - IP Range Tags

2.12.1 Overview

A L3 network is a logic network that contains a subnet and a set of network services, and that is built up on a *L2 network* that is responsible for providing isolation method. Network services, which are provided by network service providers associated with the underlying L2 network, are usually software that implement protocols spanning from OSI layer 3 to OSI layer 7.

Subnet

In a L3 network, the subnet can have a single consecutive IP range or multiple split IP ranges. The split IP ranges are typically useful when a portion of IP addresses needs to be reserved from the subnet. For example, let's say you are going to create a L3 network as a management network that has a subnet 192.168.0.0/24; however, IP addresses of 192.168.0.50 ~ 192.168.0.100 have been occupied by some network devices and you don't want ZStack to use them, then you can create two split IP ranges:

```
IP Range1

start IP: 192.168.0.2
end IP: 192.168.0.49
gateway: 192.168.0.1
netmask: 255.255.255.0

IP Range2

start IP: 192.168.0.101
end IP: 192.168.0.254
gateway: 192.168.0.1
netmask: 255.255.255.0
```

You can create split IP ranges as many as you want, as long as they all belong to the same CIDR.

Network Services

Network services implementing OSI layer 3 ~ layer 7 protocols aim to serve VMs on a L3 network. Network services are provided by network services providers that are associated to the parent L2 network of a L3 network. A type of network service can have multiple providers, and a provider can provide several types of different services. After a L3 network is created, users can attach network services to it and choose network services providers. In this ZStack version, a table of supported services/providers is shown as follows:

Network Service	Provider	Attachable L2 Network	Since
DHCP	Virtual Router	• L2NoVlanNetwork • L2VlanNetwork	0.6
DNS	Virtual Router	• L2NoVlanNetwork • L2VlanNetwork	0.6
Source NAT (SNAT)	Virtual Router	• L2NoVlanNetwork • L2VlanNetwork	0.6
Port Forwarding	Virtual Router	• L2NoVlanNetwork • L2VlanNetwork	0.6
Elastic IP (EIP)	Virtual Router	• L2NoVlanNetwork • L2VlanNetwork	0.6
Security Group	Security Group	• L2NoVlanNetwork • L2VlanNetwork	0.6

In the table, the column 'Attachable L2 Network' indicates what L2 networks providers can attach. If a provider cannot attach to a L2 network, it cannot provide services to child L3 networks of the L2 network.

2.12.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
zoneUuid	uuid of ancestor zone, see *zone*			0.6
l2NetworkUuid	uuid of parent L2 network, see *L2 network*			0.6
state	see *state*		• Enabled • Disabled	0.6
dnsDomain	see *domain*	true		0.6
ipRanges	a list of *IP ranges*			0.6
dns	a list of *DNS*			0.6
networkServices	a list of *network services references*			0.6
type	L3 network type		• L3BasicNetwork	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example


```
{
  "inventory": {
    "uuid": "f73926eb4f234f8195c61c33d8db419d",
    "name": "GuestNetwork",
    "description": "Test",
    "type": "L3BasicNetwork",
    "zoneUuid": "732fbb4383b24b019f60d862995976bf",
    "l2NetworkUuid": "f1a092c6914840c9895c564abbc55375",
    "state": "Enabled",
    "createDate": "Jun 1, 2015 11:07:24 PM",
    "lastOpDate": "Jun 1, 2015 11:07:24 PM",
    "dns": [],
    "ipRanges": [
      {
        "uuid": "78b43f4b0a9745fab49c967e1c35beb1",
        "l3NetworkUuid": "f73926eb4f234f8195c61c33d8db419d",
```

```
        "name": "TestIpRange",
        "description": "Test",
        "startIp": "10.10.2.100",
        "endIp": "10.20.2.200",
        "netmask": "255.0.0.0",
        "gateway": "10.10.2.1",
        "createDate": "Jun 1, 2015 11:07:24 PM",
        "lastOpDate": "Jun 1, 2015 11:07:24 PM"
      }
    ],
    "networkServices": [
      {
        "l3NetworkUuid": "f73926eb4f234f8195c61c33d8db419d",
        "networkServiceProviderUuid": "bbb525dc4cc8451295d379797e092dba",
        "networkServiceType": "DHCP"
      }
    ]
  }
}
```


State

L3 networks have two states:

- **Enabled**

  The state that allows new VMs to be created

- **Disabled**

  The state that DOESN'T allow new VMs to be created

Note: Existing VMs on disabled L3 networks can still be stopped, started, rebooted, and deleted.

DNS Domain

The DNS domain is used to expand hostnames of VMs on the L3 network to FQDNs(Full Qualified Domain Name); for example, if the hostname of a VM is 'vm1' and the DNS domain of the L3 network is 'zstack.org', the final hostname will be expanded to 'vm1.zstack.org'.

IP Range

In this ZStack version, only IPv4 IP range is supported.

Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
startIp	the first IP in range			0.6
endIp	the last IP in range			0.6
netmask	netmask of subnet			0.6
gateway	gateway of subnet			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example

```
{
  "inventory": {
    "uuid": "b1cfcdeca4024d13ac82edbe8d959720",
    "l3NetworkUuid": "50e637dc68b7480291ba87cbb81d94ad",
    "name": "TestIpRange",
    "description": "Test",
    "startIp": "10.0.0.100",
    "endIp": "10.10.1.200",
    "netmask": "255.0.0.0",
    "gateway": "10.0.0.1",
    "createDate": "Jun 1, 2015 4:30:23 PM",
    "lastOpDate": "Jun 1, 2015 4:30:23 PM"
  }
}
```

DNS

A L3 network can have one or more DNS that take effect when the DNS network service is enabled.

Note: In this ZStack version, only IPv4 DNS is supported

L2 Networks and L3 Networks

As a layer2 broadcast domain can contain multiple subnets, nothing will stop you from creating multiple L3 networks on the same L2 network; however, those L3 networks are not isolated and network snooping can happen; please use on your own risks.

Network Service References

Network service references exhibit network services enabled on the L3 network and their providers.

Inventory

Properties

Name	Description	Optional	Choices	Since
l3NetworkUuid	L3 network Uuid			0.6
networkServiceProviderUuid	network service provider UUID			0.6
networkServiceType	network service type		• DHCP • DNS • SNAT • PortForwarding • EIP • SecurityGroup	0.6

Example

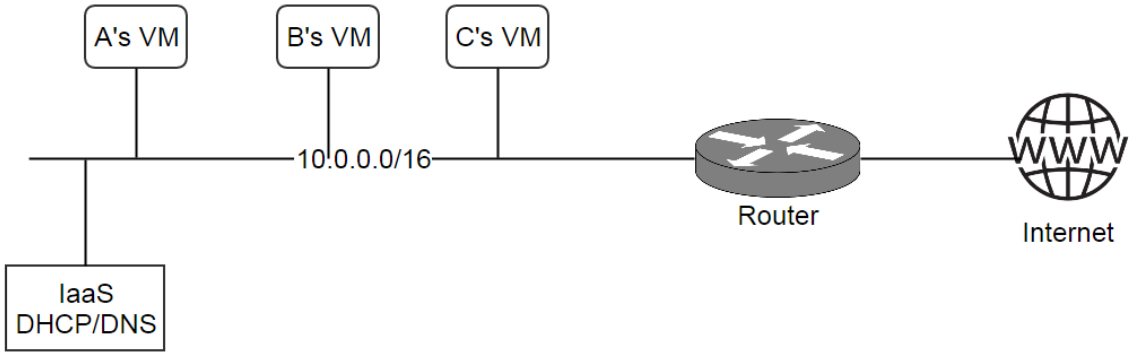
```
{
  "l3NetworkUuid": "f73926eb4f234f8195c61c33d8db419d",
  "networkServiceProviderUuid": "bbb525dc4cc8451295d379797e092dba",
  "networkServiceType": "PortForwarding"
}
```

2.12.3 Network Typology

The most common network typologies in IaaS software managed clouds are:

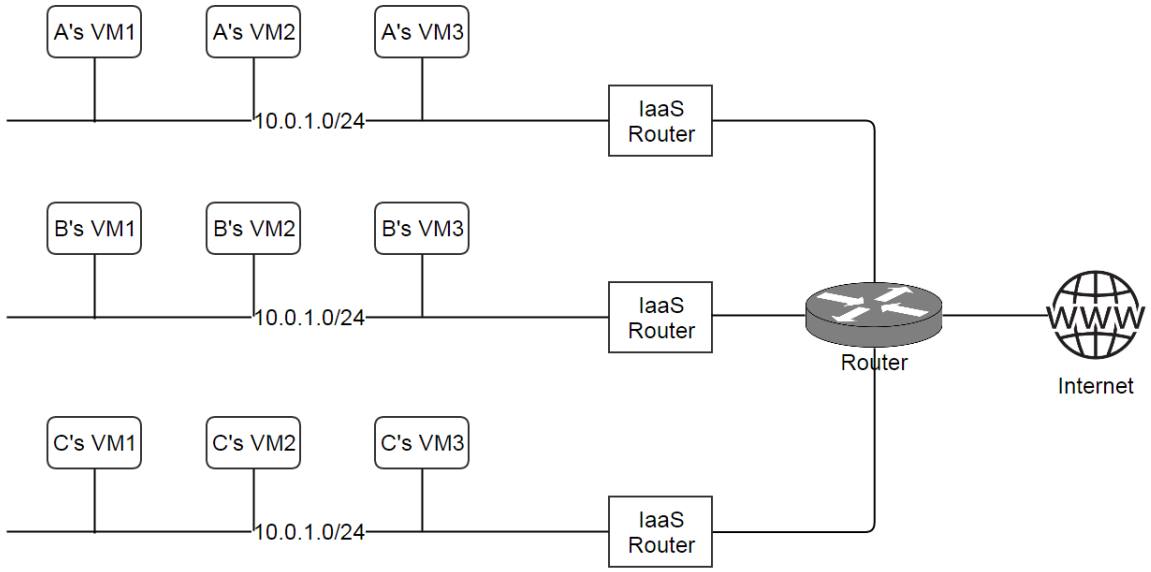
- **Flat Network or Shared Network**:

  In this typology, all tenants share a single subnet; IaaS software only provides DHCP, DNS services; the router of datacenter is responsible for routing



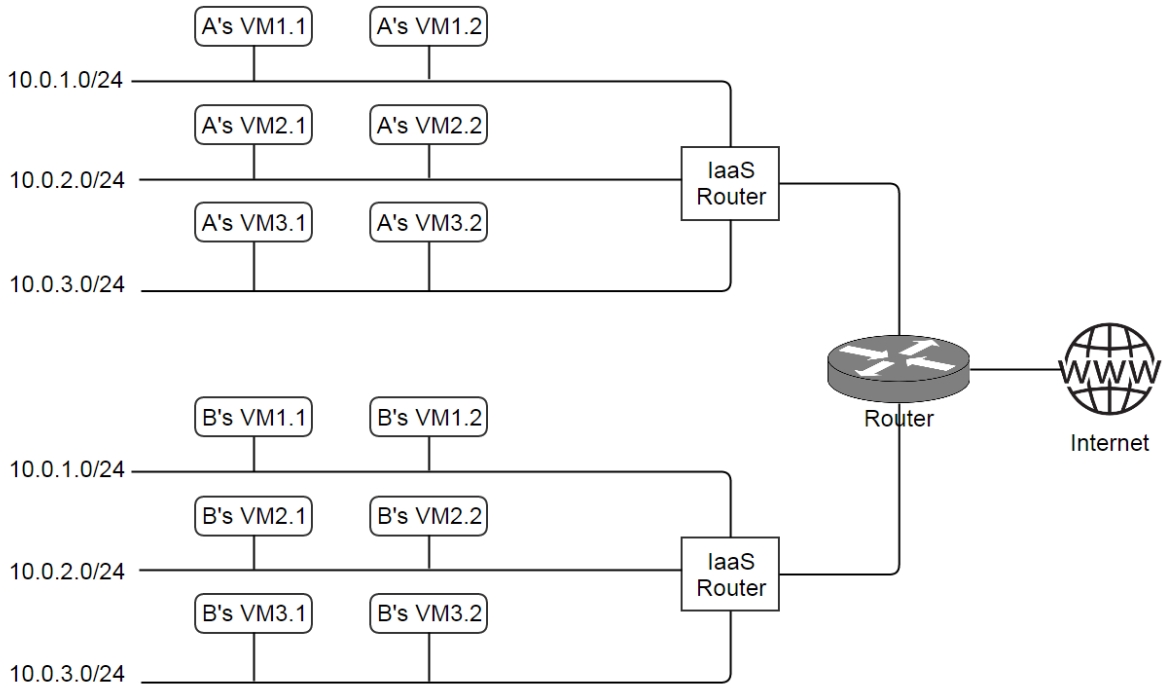
- **Private Network or Isolated Network**:

  In this typology, each tenant has own subnet; IaaS software is responsible for providing routers for all subnets, which usually have DHCP, DNS, and NAT services.

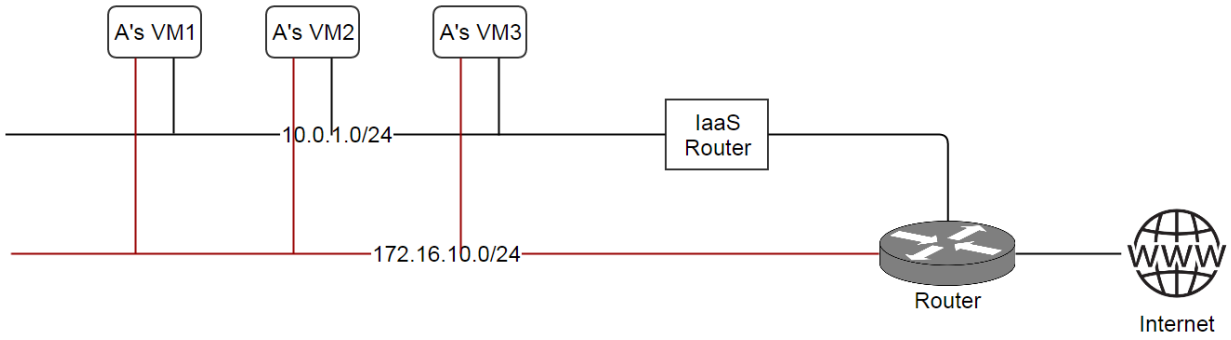


- **Virtual Private Network (VPC)**:

  In this typology, each tenant can have multiple subnets; IaaS software is responsible for providing a router coordinating all subnets; tenants can configure the routing table of the router to control connectivity amid subnets.



Besides, typical typologies can be combined to new typologies; for example, a flat network and a private network can be put together, as:



In ZStack, all those typologies can be implemented by assembling L2 networks, L3 networks and network services. For example, to create a flat network, users can create a L3 network with only DHCP, DNS enabled; to create a private network, users can create a L3 network on a L2VlanNetwork with DHCP, DNS, SNAT enabled.

Note: In this ZStack version, VPC is not supported yet.

2.12.4 Operations

Create L3 Network

Users can use CreateL3Network to create a L3 network. For example:

```
CreateL3Network l2NetworkUuid=f1a092c6914840c9895c564abbc55375 name=GuestNetwork
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
l2NetworkUuid	uuid of parent L2 network, see *L2 network*			0.6
dnsDomain	a DNS domain, see *domain*	true		0.6
type	L3 network type, see *type*	true	• L3BasicNetwork	0.6
system	indicates whether this is a system L3 network, see *System L3 Network*	true	• true • false	0.6

Type

In this ZStack version, the only L3 network type is L3BasicNetwork. Users can leave field 'type' alone when calling CreateL3Network.

System L3 Network

A system L3 network is reserved for ZStack and cannot be used to create user VMs. System L3 networks are typically used for public networks and management networks. Usually, user VMs in a cloud should not have nics on a public network and a management network, but appliance VMs (e.g router VM) do need have nics on those networks; then the management network and the public network can be created as system L3 networks.

Note: Management networks and public networks can also be created as non-system L3 networks, which allows user VMs to use them. This is normally seen in private clouds; for example, creating a user VM with a public IP directly.

Delete L3 Network

Users can use DeleteL3Network to delete a L3 network. For example:

```
DeleteL3Network uuid=f73926eb4f234f8195c61c33d8db419d
```

Parameters

Name	Description	Optional	Choices	Since
uuid	L3 network uuid			0.6
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6

Danger: Deleting a L3 network will stop all VMs that have nics on it and will delete the nics from VMs; if the nic on the L3 network is the only nic of a VM, the VM will be deleted as well. There is no way to recover a deleted L3 network.

Add IP Ranges

Add Split Ranges

Users can use AddIpRange to add an IP range to a L3 network; this is useful for adding split IP ranges. For example:

```
AddIpRange name=ipr1 startIp=192.168.0.2 endIp=192.168.0.100 netmask=255.255.255.0 gateway=192.168.0.
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
l3NetworkUuid	uuid of parent L3 network			0.6
startIp	the first IP in range			0.6
endIp	the last IP in range			0.6
netmask	netmask of subnet			0.6
gateway	gateway of subnet			0.6

Add CIDR

Users can also use AddIpRangeByNetworkCidr to add an IP range. For example:

```
AddIpRangeByNetworkCidr name=ipr1 l3NetworkUuid=50e637dc68b7480291ba87cbb81d94ad networkCidr=10.0.1.0
```

Parameters

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
l3NetworkUuid	uuid of parent L3 network			0.6
networkCidr	network CIDR; it must be in format of: `network-number/prefix-length`			0.6

Delete IP Range

Users can use DeleteIpRange to delete an IP range. For example:

```
DeleteIpRange uuid=b1cfcdeca4024d13ac82edbe8d959720
```

Warning: Deleting a IP range will stop all VMs that have IP addresses in the range. There is no way to recover a deleted IP range.

Parameters

Name	Description	Optional	Choices	Since
uuid	IP range uuid			0.6
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6

Add DNS

Users can use AddDnsToL3Network to add a DNS to a L3 network. For example:

```
AddDnsToL3Network l3NetworkUuid=50e637dc68b7480291ba87cbb81d94ad dns=8.8.8.8
```

Parameters

Name	Description	Optional	Choices	Since
l3NetworkUuid	uuid of parent L3 network			0.6
dns	dns IPv4 address			0.6

Attach Network Service

After creating a L3 network and before creating any VMs on it, users can use AttachNetworkServiceToL3Network to attach network services to the L3 network. If a network service is attached to a L3 network that already has VMs running, the existing VMs can not use the network service until they are rebooted.

Note: In this ZStack version, detaching a network service from a L3 network is not supported.

For example:

```
AttachNetworkServiceToL3Network l3NetworkUuid=50e637dc68b7480291ba87cbb81d94ad networkServices='{"1d
```

Parameters

Name	Description	Optional	Choices	Since
l3NetworkUuid	L3 network uuid			0.6
networkServices	A map whose key is network service provider UUID and value is a list of network service types			0.6

Note: You can use QueryNetworkServiceProvider to get the UUID of a network service provider, for example:

```
QueryNetworkServiceProvider fields=uuid name=VirtualRouter
```

If you want to view network services a provider provides, omit the parameter 'field', for example:

```
QueryNetworkServiceProvider name=VirtualRouter
```

Query L3 Network

Users can use QueryL3Network to query L3 networks. For example:

```
QueryL3Network dnsDomain=zstack.org
```

```
QueryL3Network vmNic.ip=192.168.10.2
```

Primitive Fields of Query

see *L3 network inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
ipRanges	*IP range inventory*	IP ranges this L3 network contains	0.6
network-Services	*l3Network network service reference*	network services attached to this L3 network	0.6
l2Network	*L2 network*	parent L2 network	0.6
vmNic	*VM nic inventory*	VM nics on this L3 network	0.6
service-Provider	network service provider inventory	network service providers that provides network services attached to this L3 network	0.6
zone	*zone inventory*	ancestor zone	0.6

2.12.5 L3 Network Tags

Users can create user tags on a L3 network with resourceType=L3NetworkVO. For example:

```
CreateUserTag resourceType=L3NetworkVO tag=web-tier-l3 resourceUuid=f6be73fa384a419986fc6d1b92f95be9
```

2.12.6 IP Range Tags

Users can create user tags on an IP range with resourceType=IpRangeVO. For example:

```
CreateUserTag resourceType=IpRangeVO tag=web-tier-IP resourceUuid=8191d946954940428b7d003166fa641e
```

2.13 Image

Table of contents

- Image
  - Overview
  - Inventory
    - Properties
    - Example
    - State
    - Status
    - URL
    - Media Type
    - Platform
    - System Image
    - Format
    - Backup Storage Reference
      - Example
  - Operations
    - Add Image
      - Parameters
    - Delete Image
      - Parameters
    - Change State
      - Parameters
    - Create RootVolumeTemplate From Root Volume
      - Parameters
      - Backup Storage UUIDs
    - Create RootVolumeTemplate From Volume Snapshot
      - Parameters
      - Backup Storage Uuids
    - Create DataVolumeTemplate From Volume
      - Parameters
      - Backup Storage Uuids
    - Query Image
      - Primitive Fields of Query
      - Nested And Expanded Fields of Query
  - Tags

2.13.1 Overview

Images provide templates for virtual machine file systems. Images can be RootVolumeTemplate that provide templates for VMs' root volumes where VMs' operating systems install; or DataVolumeTemplate that provide templates for VMs' data volumes that usually contain non operating system data; or ISO that can be used to install operating systems to blank root volumes.

Images are stored on *backup storage*. Prior to starting a VM, if the image to create VM root volume is not in *primary storage*'s image cache, it will be downloaded to the cache first. So when creating a VM with an image at the first time, it takes longer than normal because the downloading process.

ZStack uses thin provisioning to create root volumes. Root volumes from the same image share the same base in primary storage's image cache, and any changes made to the root volumes do not affect the base image.

2.13.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
state	see *state*		• Enabled • Disabled	0.6
status	see *status*		• Creating • Downloading • Ready	0.6
size	image size, in bytes			0.6
url	url the image registered from, see *url*			0.6
mediaType	image's media type, see *media type*		• RootVolumeTemplate • DataVolumeTemplate • ISO	0.6
guestOsType	a string for user records VM's operating system type	true		0.6
platform	indicates platform of VM's operating system, see *platform*		• Linux • Windows • Paravirtualization • Other	0.6
system	see *system image*			0.6
format	see *format*		• qcow2 • raw	0.6
md5Sum	image's md5sum **Note:** MD5 sum is not calculated at this ZStack version			0.6
type	reserved field		• zstack	0.6
backupStorageRefs	a list of *backup storage reference*			0.6

Example


```
{
    "backupStorageRefs": [
        {
            "backupStorageUuid": "8b99641a4d644820932e0ec5ada78eed",
            "createDate": "Jun 1, 2015 6:17:48 PM",
            "imageUuid": "b395386bdb4a4ff1b1850a457c949c5e",
            "installPath": "/export/backupStorage/sftp/templates/acct-36c27e8ff05c4780bf6d2fa65700f22
            "lastOpDate": "Jun 1, 2015 6:17:48 PM"
        }
    ],
    "createDate": "Jun 1, 2015 6:17:40 PM",
    "description": "Test Image Template for network test",
    "format": "qcow2",
    "guestOsType": "unknown",
    "lastOpDate": "Jun 1, 2015 6:17:40 PM",
    "md5Sum": "not calculated",
    "mediaType": "RootVolumeTemplate",
    "name": "image_for_sg_test",
    "platform": "Linux",
    "size": 419430400,
    "state": "Enabled",
    "status": "Ready",
    "system": false,
    "type": "zstack",
    "url": "http://172.16.0.220/templates/centos_400m_140925.img",
    "uuid": "b395386bdb4a4ff1b1850a457c949c5e"
},
```


State

Images have two states:

- **Enabled**:

  The state that allows VMs to be created from this image

- **Disabled**:

  The state that DOESN'T allow VMs to be created from this image

Status

Status indicates images' lifecycle:

- **Creating**:

  The image is in process of creating from a volume or a volume snapshot; not ready to use.

- **Downloading**:

  The image is in process of downloading from a url; not ready to use.

- **Ready**:

  The image is on backup storage and ready to use.

URL

Depending on how an image was created on a backup storage, the url has different meanings; when an image was downloaded from a web server, the url is the HTTP/HTTPS link; when an image was created from a volume or a volume snapshot, the url is a string encoding UUID of the volume or the volume snapshot, like:

```
volume://b395386bdb4a4ff1b1850a457c949c5e
volumeSnapshot://b395386bdb4a4ff1b1850a457c949c5e
```

Note: In this ZStack version, the only way to register an image to backup storage is providing a URL that is a HTTP/HTTPS link and calling AddImage.

Media Type

A media type indicates the image's usage.

- **RootVolumeTemplate**:

  The image is used to create root volumes.

- **DataVolumeTemplate**:

  The image is used to create data volumes.

- **ISO**:

  The image is used to install operating systems to blank root volumes.

Platform

Platform gives ZStack a hint that whether to use paravirtualization for VMs created from this image.

Use paravirtualization	• Linux • Paravirtualization
Not to use paravirtualization	• Windows • Other

System Image

System images are images used only for appliance VMs but not for user VMs. This is normally used for *virtual router* image in this ZStack version.

Format

Format exhibits relationships between hypervisors and images. For example, images of format qcow2 can only be used for VMs of KVM. In this ZStack version, as KVM is the only supported hypervisor, the relationship table is like:

Hypervisor Type	Format
KVM	• qcow2 • raw

Volumes will inherit formats of images from which they are created; for example, root volumes created from images of format qcow2 will have format qcow2 too. Format 'raw' is an exception, volumes created from 'raw' images will have the format qcow2 because ZStack will thin-clone it using qcow2 format.

Backup Storage Reference

An image can be stored on more than one backup storage. For every backup storage, the image has a backup storage reference encompassing backup storage UUID and image's installation path.

Name	Description	Optional	Choices	Since
imageUuid	image uuid			0.6
backupStorageUuid	backup storage uuid, see *backup storage*			0.6
installPath	installation path on backup storage			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example

```
{
    "backupStorageUuid": "8b99641a4d644820932e0ec5ada78eed",
    "imageUuid": "b395386bdb4a4ff1b1850a457c949c5e",
    "installPath": "/export/backupStorage/sftp/templates/acct-36c27e8ff05c4780bf6d2fa65700f22e/b3953
    "createDate": "Jun 1, 2015 6:17:48 PM",
    "lastOpDate": "Jun 1, 2015 6:17:48 PM"
}
```

2.13.3 Operations

Add Image

Admins can use AddImage to add an image. For example:

```
AddImage name=CentOS7 format=qcow2 backupStorageUuids=8b99641a4d644820932e0ec5ada78eed url=http://172
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
url	HTTP/HTTPS url, see *url*			0.6
mediaType	image media type, see *media type*. Default is RootVolumeTemplate	true	• RootVolumeTemplate • DataVolumeTemplate • ISO	0.6
guestOsType	a string that indicates VM's operating system type, for example, CentOS7	true		0.6
system	indicates whether this is a system image, see *system image*. Default is false	true	• true • false	0.6
format	image format, see *format*		• qcow2 • raw	0.6
platform	image platform, see *platform*. Default is Linux	true	• Linux • Windows • Other • Paravirtualization	0.6
backupStorageUuids	a list of backup storage uuid to which the image is going to add			0.6
type	reserved field, leave it alone	true	• zstack	0.6

An image can be added to multiple backup storage by providing a list of backup storage UUID in 'backupStorageUuids'; The AddImage command succeeds as long as the image is successfully added to one backup storage, and fails if the image fails on all backup storage. Backup storage that successfully added the image can be retrieved from *image backup storage reference* of the image inventory in the API response.

Delete Image

Admins can use DeleteImage to delete an image from specified backup storage or all backup storage. For example:

```
DeleteImage uuid=b395386bdb4a4ff1b1850a457c949c5e backupStorageUuids=c310386bdb4a4ff1b1850a457c949c5e
```

Parameters

Name	Description	Optional	Choices	Since
uuid	image uuid			0.6
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
backupStorageUuids	a list of backup storage storing the image; if omitted, the image will be deleted from all backup storage it's on.			0.6

An image is considered as deleted only if it is deleted from all backup storage; otherwise, its copy get deleted on some specific backup storage.

Danger: There is no way to recover an image if it has been deleted from all backup storage.

Change State

Admins can use ChangeImageState to change the state of an image. For example:

```
ChangeImageState stateEvent=enable uuid=b395386bdb4a4ff1b1850a457c949c5e
```

Parameters

Name	Description	Optional	Choices	Since
uuid	image uuid			0.6
stateEvent	state trigger event • enable: change state to Enabled • disable: change state to Disabled		• enable • disable	0.6

Create RootVolumeTemplate From Root Volume

Users can create an RootVolumeTemplate image from a root volume. For example:

```
CreateRootVolumeTemplateFromRootVolume name=CentOS7 rootVolumeUuid=1ab2386bdb4a4ff1b1850a457c949c5e
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
backupStorageUuids	a list of backup storage uuid on which the image is going to created, see *backup storage uuids*	true		0.6
rootVolumeUuid	uuid of root volume from which the image is going to create			0.6
platform	image platform, see *platform*; default to Linux	true	• Linux • Windows • Other • Paravirtualization	0.6
guestOsType	a string that indicates VM's operating system type, for example, CentOS7	true		0.6
system	indicates whether this is system image, see *system image*; default to false	true	• true • false	0.6

Backup Storage UUIDs

When calling CreateRootVolumeTemplateFromRootVolume, users can provide a list of backup storage UUIDs to specify where the image is going to create; if this field is omitted, a random backup storage will be chosen.

Create RootVolumeTemplate From Volume Snapshot

Users can use CreateRootVolumeTemplateFromVolumeSnapshot to create a RootVolumeTemplate from a volume snapshot. For example:

```
CreateRootVolumeTemplateFromVolumeSnapshot name=CentOS7 snapshotUuid=1ab2386bdb4a4ff1b1850a457c949c5e
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
snapshotUuid	uuid of volume snapshot, see *volume snapshot*			0.6
backupStorageUuids	a list of backup storage uuid on which the image is going to created, see *backup storage uuids*	true		0.6
platform	image platform, see *platform*. Default to Linux	true	• Linux • Windows • Other • Paravirtualization	0.6
guestOsType	a string that indicates VM's operating system type, for example, CentOS7	true		0.6
system	indicates whether this is system image, see *system image*. Default is false	true	• true • false	0.6

Backup Storage Uuids

When calling CreateRootVolumeTemplateFromVolumeSnapshot, users can provide a list of backup storage UUIDs to specify where the image is going to create; if this field is omitted, a random backup storage will be chosen.

Create DataVolumeTemplate From Volume

Users can use CreateDataVolumeTemplateFromVolume to create a DataVolumeTemplate from a volume. For example:

```
CreateDataVolumeTemplateFromVolume name=data volumeUuid=1ab2386bdb4a4ff1b1850a457c949c5e
```

The volume can be either root volume or data volume. This provides a way to create a data volume from a root volume. Users can firstly create a DataVolumeTemplate from a root volume, then create a data volume from the DataVolumeTemplate.

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
volumeUuid	uuid of volume, see *volume*			0.6
backupStorageUuids	a list of backup storage uuid on which the image is going to created, see *backup storage uuids*	true		0.6

Backup Storage Uuids

When calling CreateDataVolumeTemplateFromVolume, users can provide a list of backup storage UUIDs to specify where the image is going to create; if this field is omitted, a random backup storage will be chosen.

Query Image

Users can use QueryImage to query images. For example:

```
QueryImage status=Ready system=true
```

```
QueryImage volume.vmInstanceUuid=85ab231e392d4dfb86510191278e9fc3
```

Primitive Fields of Query

see *image inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
backupStorage	*backup storage inventory*	backup storage this image is on	0.6
volume	*volume inventory*	volumes created from this image	0.6
backupStorageRef	*backup storage reference*	reference used to query by backup storage install path	0.6

2.13.4 Tags

Users can create user tags on an image with resourceType=ImageVO. For example:

```
CreateUserTag resourceType=ImageVO tag=golden-image resourceUuid=ff7c04c4e2874a21a3e795501f1bc516
```

2.14 Backup Storage

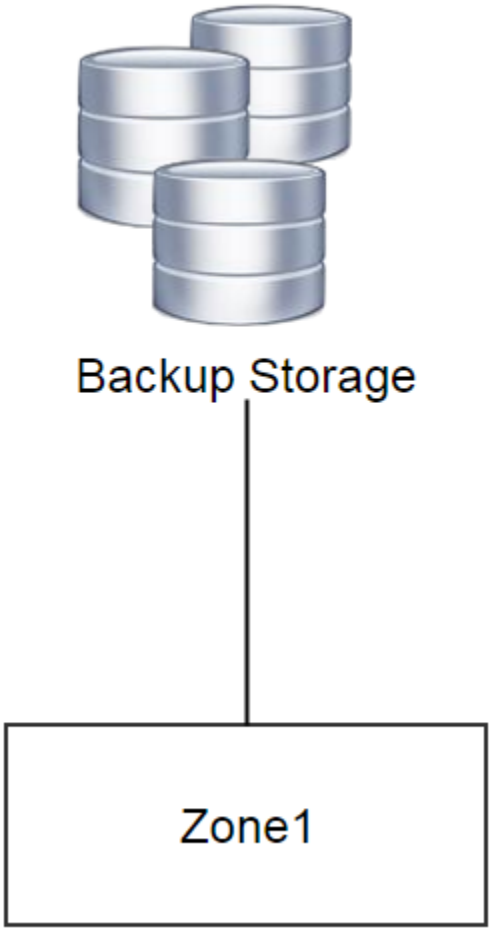
Table of contents

- *Backup Storage*
  - *Overview*
  - *Inventory*
    - *Properties*
      - *Example*
      - *URL*
      - *SFTP Backup Storage URL*
      - *Capacity*
      - *State*
      - *Status*
  - *SFTP Backup Storage*
    - *Example*
  - *Operations*
    - *Add Backup Storage*
      - *Add SFTP Backup Storage*
      - *Parameters*
    - *Delete Backup Storage*
      - *Parameters*
    - *Change State*
      - *Parameters*
    - *Attach Zone*
      - *Parameters*
    - *Detach Zone*
      - *Parameters*
    - *Query Backup Storage*
      - *Primitive Fields of Query*
      - *Nested And Expanded Fields of Query*
    - *Query SFTP Backup Storage*
      - *Primitive Fields of Query*
      - *Nested and Expanded Fields of Query*
  - *Global Configurations*
    - *ping.interval*
    - *ping.parallelismDegree*
  - *Tags*

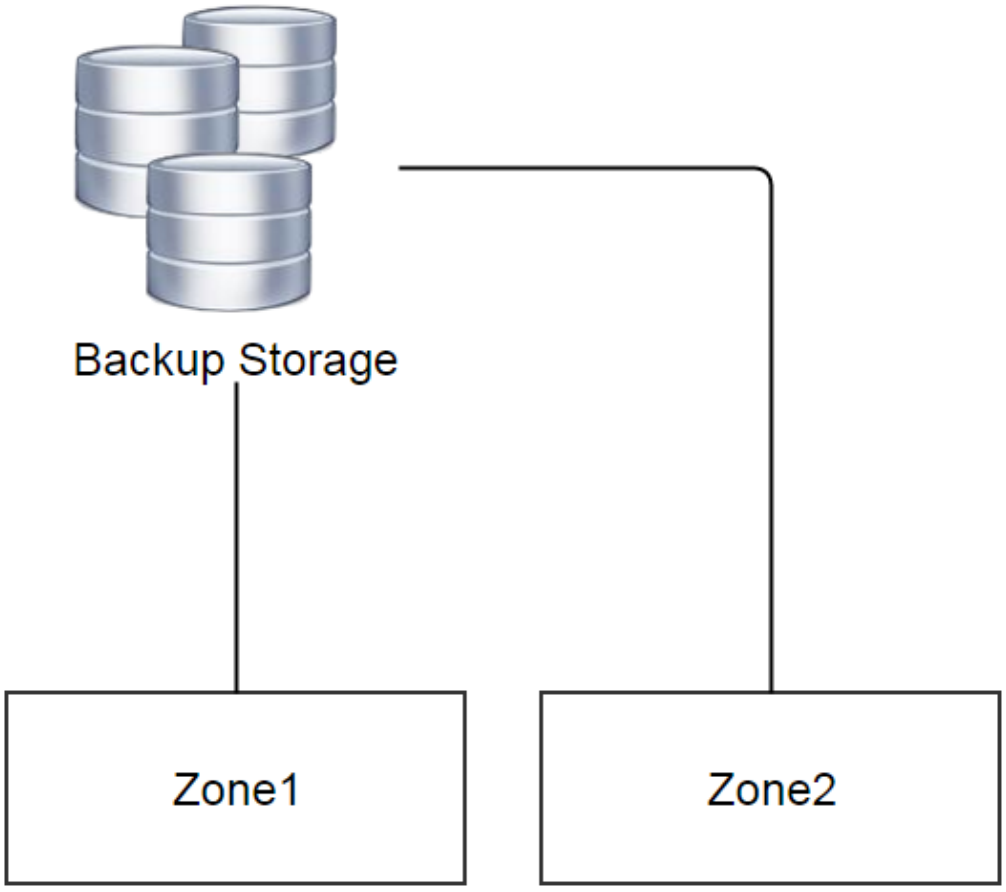
2.14.1 Overview

A backup Storage is a storage system that stores *images* for creating volumes. Backup storage can be filesystem based storage(e.g. NFS) or object store based storage(e.g. OpenStack swift), as long as the storage is network shared storage. Besides providing templates for creating volumes, backup storage also allow users to backup entities including volumes and volume snapshots.

A backup storage must be attached to a *zone* before the zone's descendant resources can access it. Admins can take this advantage to share images across multiple zones, for example:



In the early stage of a cloud, there may be only one zone(Zone1) with a single backup storage. In pace with business development, admins may decide to create another zone(Zone2) but still use existing images for VMs; then admins can attach the backup storage to Zone2, so both Zone1 and Zone2 share the same images.



Note: In this ZStack version, the only supported backup storage is *SFTP backup storage*

2.14.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
url	see *url*			0.6
totalCapacity	total disk capacity in bytes, see *capacity*			0.6
availableCapacity	available disk capacity in bytes, see *capacity*			0.6
type	backup storage type		• SftpBackupStorage	0.6
state	see *state*		• Enabled • Disabled	0.6
status	see *status*		• Connecting • Connected • Disconnected	0.6
attachedZoneUuids	a list of zone UUID the backup storage has been attached			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example

```
{
    "attachedZoneUuids": [
        "36de66d82f424639af67215a465418f6"
    ],
    "availableCapacity": 1258407346176,
    "name": "sftp",
    "state": "Enabled",
    "status": "Connected",
    "totalCapacity": 1585341214720,
    "type": "SftpBackupStorage",
    "url": "/export/backupStorage/sftp",
    "uuid": "33a35f75885f45ab96ea2626ce9c05a6",
    "lastOpDate": "Jun 1, 2015 3:42:26 PM",
    "createDate": "Jun 1, 2015 3:42:26 PM"
}
```

URL

URL is a string that contains information needed by backup storage plugins for manipulating storage systems. Although it's named as URL, the certain format of the string is up to backup storage types and is not necessary to strictly follow the URL convention, to give flexibilities to plugins to encode information that may not be able to fit in the URL format.

SFTP Backup Storage URL For SFTP backup storage, the URL is the absolute path of a directory in the filesystem. For example, /storage/sftp.

Capacity

ZStack keeps tracking disk capacities of backup storage in order to select suitable one when allocating space for storing images. The capacity is calculated by below formulas:

```
totalCapacity = backup storage's total capacity
availableCapacity = totalCapacity - sum(images' real sizes)
```

State

Backup storage have two states:

- **Enabled**:

  The state that allows images to be registered, backup, and downloaded

- **Disabled**:

  The state that DOESN'T allow images to be registered, backup, and downloaded. Especially, if an image is only stored on a disabled backup storage, and if that image is not downloaded to image caches of primary storage yet, no VMs can be created from that image.

Status

Status reflects the status of command channels amid ZStack management nodes and backup storage.

- **Connecting**:

  A ZStack management node is trying to establish the command channel between itself and a backup storage. No operations can be performed to the backup storage.

- **Connected**

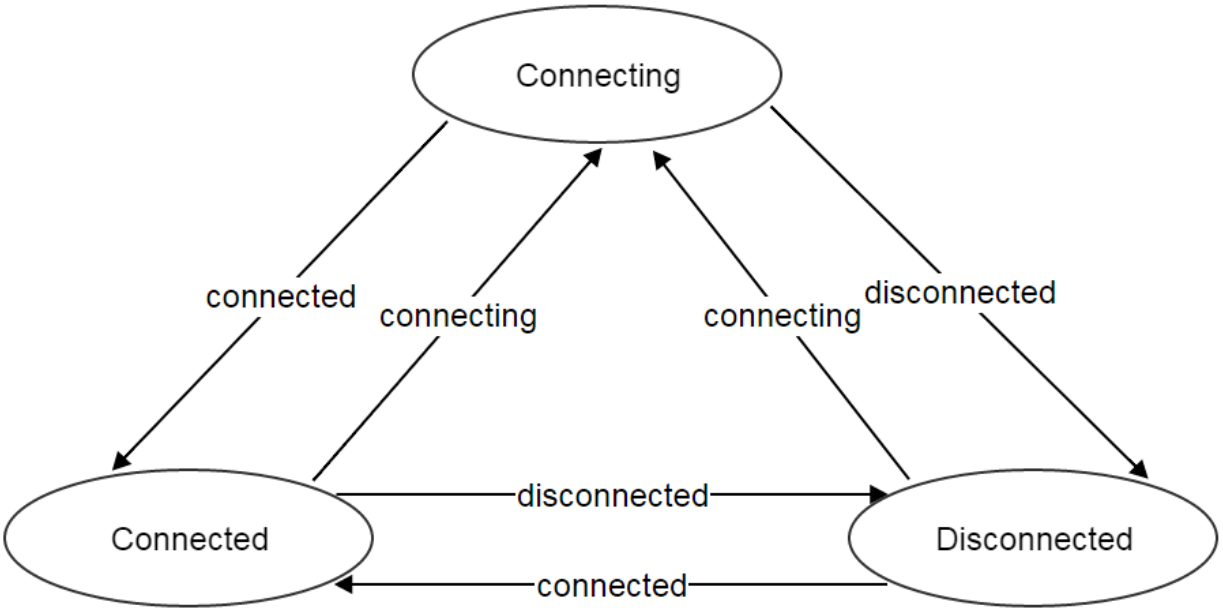
  The command channel has been successfully established between a ZStack management node and a backup storage. Operations can be performed to the backup storage.

- **Disconnected**

  The command channel has lost between a ZStack management node and a backup storage. No operations can be performed to the backup storage.

ZStack management nodes will try to setup command channels every time when they boot, and will periodically send ping commands to backup storage to check the health of command channels. Once a backup storage fails to respond, or a ping command times out, the command channel is considered as lost and the backup storage will be placed in the status of Disconnected.

Here is the transition diagram:



2.14.3 SFTP Backup Storage

SFTP backup storage is a Linux server that stores images in native filesystem and uses OpenSSH server/client to transfer images. ZStack uses a python agent (SftpBackupStorageAgent) to manage the Linux server; images are uploaded/downloaded to/from the server by SCP. Besides properties in *backup storage inventory*, SFTP backup storage has an extra property:

Name	Description	Optional	Choices	Since
hostname	the IP address or DNS name of the SFTP backup storage			0.6

Example


```
{
    "attachedZoneUuids": [
        "36de66d82f424639af67215a465418f6"
    ],
    "availableCapacity": 1258407346176,
    "hostname": "172.16.0.220",
    "name": "sftp",
    "state": "Enabled",
    "status": "Connected",
    "totalCapacity": 1585341214720,
    "type": "SftpBackupStorage",
    "url": "/export/backupStorage/sftp",
    "uuid": "33a35f75885f45ab96ea2626ce9c05a6",
    "lastOpDate": "Jun 1, 2015 3:42:26 PM",
```

```
    "createDate": "Jun 1, 2015 3:42:26 PM"
}
```

2.14.4 Operations

Add Backup Storage

The commands to add a backup storage vary for different backup storage types.

Add SFTP Backup Storage

Admins can use AddSftpBackupStorage to add a new backup storage. For example:

```
AddSftpBackupStorage name=sftp1 url=/storage/sftp1 hostname=192.168.0.220 username=root password=pass
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
url	see *url*			0.6
hostname	the IP address or DNS name of the SFTP backup storage			0.6
username	the user **root**		root	0.6
password	the SSH password for user **root**			0.6

Delete Backup Storage

Admins can use DeleteBackupStorage to delete a backup storage. For example:

```
DeleteBackupStorage uuid=1613b627cb2e4ffcb30e7e59935064be
```

Warning: When deleting, a backup storage will be detached from attached zones. Copies of images and of volume snapshots on the backup storage will be deleted; if a copy is the only copy of an image or a volume snapshot, the image or the volume snapshot will be deleted as well. There is no way to recover a deleted backup storage.

Parameters

Name	Description	Optional	Choices	Since
uuid	backup storage uuid			0.6
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6

Change State

Admins can use ChangeBackupStorageState to change the state of a backup storage. For example:

```
ChangeBackupStorageState uuid=33a35f75885f45ab96ea2626ce9c05a6 stateEvent=enable
```

Parameters

Name	Description	Optional	Choices	Since
uuid	backup storage uuid			0.6
stateEvent	state trigger event • enable: change the state to Enabled • disable: change the state to Disabled		• enable • disable	0.6

Attach Zone

Admins can use AttachBackupStorageToZone to attach a backup storage to a zone. For example:

```
AttachBackupStorageToZone backupStorageUuid=d086c30f33914c98a6078269bab7bc8f zoneUuid=d086c30f33914c9
```

Parameters

Name	Description	Optional	Choices	Since
backupStorageUuid	the backup storage uuid			0.6
zoneUuid	the zone uuid			0.6

Detach Zone

Admins can use DetachBackupStorageFromZone to detach a backup storage from a zone. For example:

```
DetachBackupStorageFromZone backupStorageUuid=d086c30f33914c98a6078269bab7bc8f zoneUuid=d086c30f3391
```

Parameters

Name	Description	Optional	Choices	Since
backupStorageUuid	the backup storage uuid			0.6
zoneUuid	the zone uuid			0.6

Query Backup Storage

Admins can use QueryBackupStorage to query backup storage. For example:

```
QueryBackupStorage state=Enabled
```

```
QueryBackupStorage image.platform=Linux
```

Primitive Fields of Query

see *backup storage inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
zone	*zone inventory*	zones this backup storage is attached to	0.6
image	*image inventory*	images this backup storage contains	0.6
volumeSnapshot	*volume snapshot inventory*	volume snapshots this backup storage contains	0.6

Query SFTP Backup Storage

Admins can use QuerySftpBackupStorage to query SFTP backup storage:

```
QuerySftpBackupStorage name=sftp
```

Primitive Fields of Query

see *SFTP backup storage inventory*

Nested and Expanded Fields of Query

see *backup storage nested and expanded fields*

2.14.5 Global Configurations

ping.interval

Name	Category	Default Value	Choices
ping.interval	backupStorage	60	> 0

The interval that management nodes send ping commands to backup storage, in seconds.

ping.parallelismDegree

Name	Category	Default Value	Choices
ping.parallelismDegree	backupStorage	50	> 0

The max number of backup storage that management nodes will ping in parallel.

2.14.6 Tags

Admins can create user tags on a backup storage with resourceType=BackupStorageVO. For example:

```
CreateUserTag tag=lab1 resourceType=BackupStorageVO resourceUuid=2906471068802c501773d3ee55b7766e
```

2.15 Volume

Table of contents

- Volume
  - Overview
  - Inventory
    - Properties
      - Example
      - Attached VM
      - Format
      - Device ID
      - State
      - Status
  - Operations
    - Create a Data Volume
      - From a Disk Offering
      - Parameters
      - From an Image
      - Parameters
      - From a Volume Snapshot
      - Parameters
    - Delete Data Volume
      - Parameters
    - Change State
      - Parameters
    - Attach VM
      - Parameters
    - Detach VM
      - Parameters
    - Query Volume
      - Primitive Fields of Query
      - Nested And Expanded Fields of Query
  - Tags

2.15.1 Overview

Volumes provide storage to guest VMs. A volume can be of type of root or data, depending on the role it plays. A root volume is a disk where the VM operating system is installed, for example, C: or sda; a data volume which provides additional storage is like an extra hard drive, for example: D: or sdb.

Volumes are hypervisor specific; that is to say, a volume created for one hypervisor may not be able to get attached to VMs of other hypervisor types; for example, a volume for KVM VMs cannot be attached to VMWare VMs. The hypervisor attribute of volumes is implied by the field *format*, which is similar to *image format* except the image format has an extra value 'ISO' that volumes don't have.

Because of thin provisioning, a volume can has two sizes: real size and virtual size. The real size is the size that a volume actually occupies in storage system; the virtual size is the size a volume claims for, which is the max size a volume can have when it is fully filled. The virtual size is always greater than or equal to the real size.

Volumes on *primary storage* can be directly accessed by VMs. A volume can only be attached to one VM at any given time. A root volume is always attached to its owner VM and cannot be detached; a data volume, on the contrary, can

be attached/detached to/from different VMs of the same hypervisor type, as long as the VMs can access the primary storage on which the data volume locates.

2.15.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
primaryStorageUuid	the uuid of primary storage the volume locates on, see *primary storage*			0.6
vmInstanceUuid	uuid of the VM the volume is attached, or NULL if not attached; see *attach VM*	true		0.6
diskOfferingUuid	the uuid of *disk offering*, if the volume is created from a disk offering	true		0.6
rootImageUuid	the uuid of *image*, if the volume is created from an image	true		0.6
installPath	the path where the volume is installed on the primary storage			0.6
type	volume type		• Root • Data	0.6
format	see *format*		• qcow2	0.6
size	the volume's virtual size, in bytes			0.6
deviceId	see *device id*	true		0.6
state	see *state*		• Enabled • Disabled	0.6
status	see *status*		• Creating • Ready • NotInstantiated	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example


```
{
    "description": "Root volume for VM[uuid:1a2b197060eb4593bf5bbf2a83b3d625]",
    "deviceId": 0,
    "format": "qcow2",
    "installPath": "/opt/zstack/nfsprimarystorage/prim-302055ec45794423af7f5d3c5081bc87/rootVolumes/a
    "createDate": "Jun 1, 2015 3:45:44 PM",
    "lastOpDate": "Jun 1, 2015 3:45:44 PM",
    "name": "ROOT-for-virtualRouter.13.1b7f47f5350c488c99e8f54142ddffbd",
    "primaryStorageUuid": "302055ec45794423af7f5d3c5081bc87",
    "rootImageUuid": "178c662bfcdd4145920682c58ebcbed4",
    "size": 1364197376,
    "state": "Enabled",
    "status": "Ready",
    "type": "Root",
    "uuid": "f7bbb3ae1c674ecda3b0f4c025e333f9",
    "vmInstanceUuid": "1a2b197060eb4593bf5bbf2a83b3d625"
}
```


Attached VM

A data volume can be attached to a Running or Stopped VM, but can only be attached to one VM at any given time; after being attached, the VM's UUID is shown up in the field 'vmInstanceUuid'. A data volume can also be detached from one VM and be re-attached to another VM, as long as VMs are of the same hypervisor type. A root volume is always attached to its owner VM and can never be detached.

Format

Format reveals relationship between a volume and a hypervisor type, indicating what VMs of which hypervisor type a volume can be attached. Volume format is similar to *image format*. In this ZStack version, as KVM is the only supported hypervisor type, the only volume format is 'qcow2'.

Device ID

Device ID shows the order that volumes are attached to a VM. Because the root volume is always the first volume attached, it has a fixed device ID 0; data volumes may have device IDs 1, 2, 3 ... N, depending on the sequence they are attached to the VM. The device ID can be used to identify the disk letter of the volume in guest operating system; for example, in Linux, 0 usually means /dev/xvda, 1 usually means /dev/xvdb and so fourth.

State

Volumes have two states:

- **Enabled**:

  The state that allows volumes to be attached to VMs.

- **Disabled**:

  The state that DOESN't allow volumes to be attached to VMs; however, an attached data volume can always be detached even if in state of Disabled.

Note: Root volumes always have the state of Enabled as they cannot be detached.

Status

Status shows lifecycle of volumes:

- **NotInstantiated**:

  A specific status for only data volumes. Data volumes of this status are only allocated in database and have not been instantiated on any primary storage yet; that is to say, they are just database records. Data volumes in status of NotInstantiated can be attached to VMs of any hypervisor types; and will be instantiated to concrete binaries on primary storage, with hypervisor types of VMs they are being attached. After being attached, data volumes' hypervisorType fields will be evaluated to hypervisor types of VMs, status will be changed to Ready; and since then they can only be re-attached to VMs of the same hypervisor types.

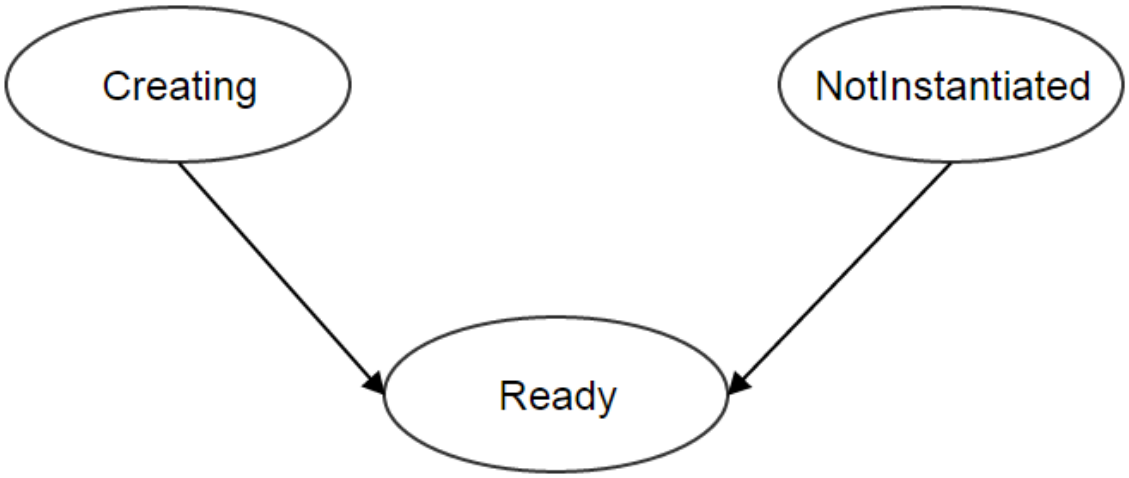
- **Ready**:

  Volumes are already instantiated on primary storage and are ready for operations.

- **Creating**:

  Volumes are in process of being created from images or volume snapshots; not ready for operations.

The status transition diagram is like:



Note: Root volume is always in status of Ready.

2.15.3 Operations

Create a Data Volume

Note: Root volumes are created automatically when creating VMs; there is no API to create root volumes.

From a Disk Offering

Users can use CreateDataVolume to create a data volume from a *disk offering*. For example:

```
CreateDataVolume name=data1 diskOfferingUuid=fea135f1d1de40b4915a19aa155983b3
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
diskOfferingUuid	disk offering uuid, see *disk offering*			0.6

From an Image

Users can use CreateDataVolumeFromVolumeTemplate to create a data volume from an image. For example:

```
CreateDataVolumeFromVolumeTemplate name=data1 imageUuid=ee6fa27ade8c42a2bdda8f9b1eee8c93 primaryStora
```

The image can be of media type of RootVolumeTemplate or DataVolumeTemplate.

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
imageUuid	image uuid, see *image*			0.6
primaryStorageUuid	uuid of primary storage where the data volume is going to be created; the primary storage must be accessible to VMs that the data volume is planned to be attached; otherwise you may create a dangling data volume that cannot be attached to VMs you want. see *primary storage*.			0.6

From a Volume Snapshot

Users can use CreateDataVolumeFromVolumeSnapshot to create a data volume from a *volume snapshot*. For example:

```
CreateDataVolumeFromVolumeSnapshot name=data1 primaryStorageUuid=302055ec45794423af7f5d3c5081bc87 vol
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
volumeSnapshotUuid	volume snapshot uuid, see *volume snapshot*			0.6
primaryStorageUuid	uuid of primary storage where the data volume is going to be created; the primary storage must be accessible to VMs that the data volume is planned to be attached; otherwise you may create a dangling data volume that cannot be attached to VMs you want. see *primary storage*.			0.6

Delete Data Volume

Users can use DeleteDataVolume to delete a data volume. For example:

```
DeleteDataVolume uuid=178c662bfcdd4145920682c58ebcbed4
```

Note: Root volumes, which are deleted when deleting VMs, cannot be deleted by APIs.

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	volume uuid			0.6

Danger: There is no way to recover a deleted data volume.

Change State

Users can use ChangeVolumeState to change the state of a data volume. For example:

```
ChangeVolumeState uuid=be19ce415bbe44539b0bd276633470e0 stateEvent=enable
```

Note: States of root volumes are unchangeable.

Parameters

Name	Description	Optional	Choices	Since
uuid	volume uuid			0.6
stateEvent	state trigger event • enable: change the state to Enabled • disable: change the state ot Disabled		• enable • disable	0.6

Attach VM

Users can use AttachDataVolumeToVm to attach a data volume to a VM. For example:

```
AttachDataVolumeToVm volumeUuid=178c662bfcdd4145920682c58ebcbed4 vmInstanceUuid=c5b443a20341418b9120c
```

Parameters

Name	Description	Optional	Choices	Since
volumeUuid	volume uuid			0.6
vmInstanceUuid	VM uuid, see *VM*			0.6

Detach VM

Users can use DetachDataVolumeFromVm to detach a data volume from a VM. For example:

```
DetachDataVolumeFromVm uuid=178c662bfcdd4145920682c58ebcbed4
```

Parameters

Name	Description	Optional	Choices	Since
uuid	volume uuid			0.6

Warning: Please flush all changes in VM operating system to disk before detaching a data volume and make sure no applications are accessing it; otherwise data in the data volume may crash. Imagine the process of detaching a data volume as hot unplugging a hard drive from a computer.

Query Volume

Users can use QueryVolume to query volumes. For example:

```
QueryVolume type=Data vmInstanceUuid=71f5376ef53a46a9abddd59c942cf45f
```

```
QueryVolume diskOffering.name=small primaryStorage.uuid=8db7eb2ccdab4c4eb4784e46895bb016
```

Primitive Fields of Query

see *volume inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
vmInstance	*VM inventory*	the VM the volume is attached to	0.6
snapshot	*volume snapshot inventory*	volume snapshots that are created from this volume	0.6
diskOffering	*disk offering inventory*	disk offering that the volume is created from	0.6
primaryStorage	*primary storage inventory*	primary storage that the volume is on	0.6
image	*image inventory*	image that the volume is create from	0.6

2.15.4 Tags

Users can create user tags on a volume with resourceType=VolumeVO. For example:

```
CreateUserTag resourceType=VolumeVO tag=goldenVolume resourceUuid=f97b8cb9bccc4872a723c8b7785d9a12
```

2.16 Disk Offering

Table of contents

- *Disk Offering*
  - *Overview*
  - *Inventory*
    - *Properties*
      - *Disk Size*
      - *State*
      - *Allocator Strategy*
  - *Operations*
    - *Create Disk Offering*
      - *Parameters*
    - *Change State*
      - *Parameters*
    - *Delete Disk Offering*
      - *Parameters*
    - *Query Disk Offering*
      - *Primitive Fields of Query*
      - *Nested And Expanded Fields of Query*
  - *Tags*
    - *System Tags*
      - *Dedicated Primary Storage*

2.16.1 Overview

A disk offering is a specification of a volume, which defines a volume's size and how it will be created. Disk offerings can be used to create both root volumes and data volumes.

Note: There is no API to create a root volume; but if you provision a VM with an ISO image, you need to specify a disk offering that defines size and allocator strategy for the VM's root volume, which is the only way that creates a root volume from a disk offering.

2.16.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
diskSize	the size of volume in bytes, see *disk size*			0.6
state	see *state*		• Enabled • Disabled	0.6
type	reserved field		• zstack	0.6
allocatorStrategy	see *allocator strategy*		• DefaultPrimaryStorageAllocationStrategy	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Disk Size

DiskSize defines a volume's virtual size. As mentioned in *volume*, virtual size is the max size a volume can occupy in storage system after it is fully filled. Putting in a straight way, it's the size you want for the volume.

State

Disk offerings have two states:

- **Enabled**:

  The state that allows volumes to be created from this disk offering

- **Disabled**:

  The state that DOESN'T allow volumes to be created from this disk offering

Allocator Strategy

Allocator strategy defines how ZStack selects a primary storage when creating a new volume. Currently the only supported strategy is DefaultPrimaryStorageAllocationStrategy that finds a primary storage satisfying conditions:

```
1. state is Enabled
2. status is Connected
3. availableCapacity is greater than disk offering's diskSize
4. has been attached to the cluster that runs the VM to which the volume will be attached
```

Note: A volume created from a disk offering is only instantiated on primary storage when it's being attached to a VM. See *volume status NotInstantiated*.

2.16.3 Operations

Create Disk Offering

Users can use CreateDiskOffering create a disk offering. For example:

```
CreateDiskOffering name=small diskSize=1073741824
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
diskSize	disk size in bytes, see *size*			0.6
allocationStrategy	see *allocator strategy*	true	• DefaultPrimaryStorageAllocationStrategy	0.6
type	reserved filed, leave it alone	true		0.6

Change State

Users can use ChangeDiskOfferingState to change the state of a disk offering. For example:

```
ChangeDiskOfferingState uuid=178c662bfcdd4145920682c58ebcbed4 stateEvent=enable
```

Parameters

Name	Description	Optional	Choices	Since
uuid	disk offering uuid			0.6
stateEvent	state trigger event • enable: change state to Enabled • disable: change state to Disabled		• enable • disable	0.6

Delete Disk Offering

Users can use DeleteDiskOffering to delete a disk offering. For example:

```
DeleteDiskOffering uuid=178c662bfcdd4145920682c58ebcbed4
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	disk offering uuid			0.6

Query Disk Offering

Users can use QueryDiskOffering to query disk offerings. For example:

```
QueryDiskOffering diskSize>=10000000
```

```
QueryDiskOffering volume.name=data1
```

Primitive Fields of Query

see *disk offering inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
volume	*volume inventory*	volumes that are created from this disk offering	0.6

2.16.4 Tags

Users can create user tags on a disk offering with resourceType=DiskOfferingVO. For example:

```
CreateUserTag tag=smallDisk resourceType=DiskOfferingVO resourceUuid=d6c49e73927d40abbfcf13852dc1836
```

System Tags

Dedicated Primary Storage

When creating volumes from disk offerings, users can use a system tag to specify primary storage on which the volumes will be created.

Tag	Description	Example	Since
primaryStorage::allocator::uuid::{uuid}	if present, volumes created from this disk offering will be allocated on the primary storage of *uuid*; an allocation failure will be raised if the specified primary storage doesn't exist or doesn't have enough capacity.	primaryStorage::allocator::uuid::68398e8b7ff24527a3b81dc4bc64d974	0.6
primaryStorage::allocator::userTag::{tag}::required	if present, volumes created from this disk offering will be allocated on the primary storage having user tag *tag*; an allocation failure will be raised if no primary storage has the *tag* or primary storage having the *tag* doesn't have enough capacity.	primaryStorage::allocator::userTag::SSD::required	0.6
primaryStorage::allocator::userTag::{tag}	if present, volumes created from this disk offering will be primarily allocated on the primary storage having user tag *tag*, if there is any; no failure will be raised if no primary storage has the *tag* or primary storage having the *tag* doesn't have enough capacity, instead, a random primary storage will be chosen for the volume.	primaryStorage::allocator::userTag::SSD	0.6

if more than one above system tags present on a disk offering, the precedent order is:

```
primaryStorage::allocator::uuid::{uuid} > primaryStorage::allocator::userTag::{tag}::required > prima
```

2.17 Instance Offering

Table of contents

- *Instance Offering*
  - *Overview*
  - *Inventory*
    - *Properties*
      - *CPU Capacity*
      - *KVM CPU Speed*
      - *Type*
      - *Allocator Strategy*
      - *DefaultHostAllocatorStrategy*
      - *Input Parameters*
      - *Algorithm*
      - *DesignatedHostAllocatorStrategy*
      - *Input Parameters*
      - *Algorithm*
      - *State*
  - *Operations*
    - *Create Instance Offering*
      - *Parameters*
    - *Delete Instance Offering*
      - *Parameters*
    - *Change State*
      - *Parameters*
    - *Query Instance Offering*
      - *Primitive Fields of Query*
      - *Nested and Expanded Fields of Query*
  - *Tags*
    - *System Tags*
      - *Dedicated Primary Storage*

2.17.1 Overview

An instance offering is a specification of VM's memory, CPU, and host allocation algorithm; it defines the volume of computing resource a VM can have.

2.17.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
cpuNum	VCPU number, see *CPU capacity*			0.6
cpuSpeed	VCPU speed, see *CPU capacity*			0.6
memorySize	memory size, in bytes			0.6
type	instance offering type, default is UserVm, see *type*	true	• UserVm • VirtualRouter	0.6
allocatorStrategy	host allocator strategy, see *allocator strategy*		• DefaultHostAllocatorStrategy • DesignatedHostAllocatorStrategy	0.6
state	see *state*		• Enabled • Disabled	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

CPU Capacity

Instance offerings use cpuNum and cpuSpeed to define a VM's CPU capacity. cpuNum, very straightforward, means the number of VCPU that a VM has; cpuSpeed is a little special; as a VM's VCPU always has the frequency same to the host's physical CPU, cpuSpeed here actually means VCPU weight in hypervisors. Depending on hypervisor types, the use and implementation of cpuSpeed vary.

KVM CPU Speed In KVM, ZStack will set the result of 'cpuSpeed * cpuNum' to VM's XML configuration to libvirt:

```
<cputune>
  <shares>128</shares>
</cputune>

shares = cpuNum * cpuSpeed
```

Type

The type of instance offering; currently there are two types:

- **UserVm**: instance offering for creating user VMs.
- **VirtualRouter**: instance offering for creating virtual router VMs; see *virtual router*.

Allocator Strategy

Allocator strategy defines the algorithm of selecting destination hosts for creating VMs.

DefaultHostAllocatorStrategy DefaultHostAllocatorStrategy uses below algorithm:

Input Parameters

Name	Description
image	image used to create the VM
L3 network	L3 networks the VM will have nics on
instance offering	instance offering
tags	tags for host allocation

Algorithm

```
l2_networks = get_parent_l2_networks(l3_networks)
host_set1 = find_hosts_in_cluster_that_have_attached_to_l2_networks()
check_if_backup_storage_having_image_have_attached_to_zone_of_hosts(host_set1)
host_set2 = remove_hosts_not_having_state_Enabled_and_status_Connected(host_set1)
host_set3 = remove_hosts_not_having_capacity_required_by_instance_offering(host_set2)
primary_storage = find_Enabled_Connected_primary_storage_having_enough_capacity_for_root_volume_and_a
host_set4 = remove_hosts_that_cannot_access_primary_storage(host_set3)
host_set5 = remove_avoided_hosts(host_set4)
host_set6 = call_tag_plugin(tags, host_set5)

return randomly_pick_one_host(host_set6)
```

DesignatedHostAllocatorStrategy DesignatedHostAllocatorStrategy uses algorithm:

Input Parameters

Name	Description	Optional
image	image used to create the VM	
L3 network	L3 networks the VM will have nics on	
instance offering	instance offering	
tags	tags for host allocation	
zone	the zone the VM wants to run	true
cluster	the cluster the VM wants to run	true
host	the host the VM wants to run	true

Algorithm

```
l2_networks = get_parent_l2_networks(l3_networks)
host_set1 = find_hosts_in_cluster_that_have_attached_to_l2_networks()
check_if_backup_storage_having_image_have_attached_to_zone_of_hosts(host_set1)

if host is not null:
```

```
   host_set2 = list(find_host_in_host_set1(host))
else if cluster is not null:
   host_set2 = find_host_in_cluster_and_host_set1(cluster)
else if zone is not null:
   host_set2 = find_host_in_zone_and_host_set1(zone)

host_set3 = remove_hosts_not_having_state_Enabled_and_status_Connected(host_set2)
host_set4 = remove_hosts_not_having_capacity_required_by_instance_offering(host_set3)
primary_storage = find_Enabled_Connected_primary_storage_having_enough_capacity_for_root_volume_and_a
host_set5 = remove_hosts_that_cannot_access_primary_storage(host_set4)
host_set6 = remove_avoided_hosts(host_set5)
host_set7 = call_tag_plugin(tags, host_set6)

return randomly_pick_one_host(host_set7)
```

Note: DesignatedHostAllocatorStrategy is a little special of not being specified in instance offerings; when a zoneUuid or a clusterUuid or a hostUuid is specified in *CreateVmInstance*, DesignatedHostAllocatorStrategy automatically overrides the strategy in instance offering.

State

Instance offerings have two states:

- **Enabled**:

  The state that allows VMs to be created from this instance offering

- **Disabled**:

  The state that DOESN'T allows VMs to be created from this instance offering

2.17.3 Operations

Create Instance Offering

Users can use CreateInstanceOffering to create an instance offering. For example:

```
CreateInstanceOffering name=small cpuNum=1 cpuSpeed=1000 memorySize=1073741824
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
cpuNum	VCPU num, see *CPU capacity*			0.6
cpuSpeed	VCPU speed, see *CPU capacity*			0.6
memorySize	memory size, in bytes			0.6
type	type, default is UserVm, see *type*	true	• UserVm • VirtualRouter	0.6

Delete Instance Offering

Users can use DeleteInstanceOffering to delete an instance offering. For example:

```
DeleteInstanceOffering uuid=1164a094fea34f1e8265c802a8048bae
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	instance offering uuid			0.6

Change State

Users can use ChangeInstanceOfferingState to change a state of instance offering. For example:

```
ChangeInstanceOfferingState uuid=1164a094fea34f1e8265c802a8048bae stateEvent=enable
```

Parameters

Name	Description	Optional	Choices	Since
stateEvent	state trigger event • enable: change state to Enabled • disable: change state to Disabled		• enable • disable	0.6
uuid	instance offering uuid			0.6

Query Instance Offering

Users can use QueryInstanceOffering to query instance offerings. For example:

```
QueryInstanceOffering cpuSpeed=512 cpuNum>2
```

```
QueryInstanceOffering vmInstance.state=Stopped
```

Primitive Fields of Query

see *instance offering inventory*

Nested and Expanded Fields of Query

Field	Inventory	Description	Since
vmInstance	*VM inventory*	VMs that are created from this instance offering	0.6

2.17.4 Tags

Users can create user tags on an instance offering with resourceType=InstanceOfferingVO. For example:

```
CreateUserTag resourceType=InstanceOfferingVO tag=web-server-offering resourceUuid=45f909969ce24865b1
```

System Tags

Dedicated Primary Storage

When creating VMs, users can use a system to specify primary storage on which root volumes will be created.

Tag	Description	Example	Since
primaryStorage::allocator::uuid::{uuid}	if present, the VM's root volume will be allocated on the primary storage whose uuid is *uuid*; an allocation failure will be raised if the specified primary storage doesn't exist or doesn't have enough capacity.	primaryStorage::allocator::uuid::68398e8b7ff24527a3b81dc4bc64d974	0.6
primaryStorage::allocator::userTag::{tag}::required	if present, the VM's root volume will be allocated on the primary storage which has user tag *tag*; an allocation failure will be raised if no primary storage has the *tag* or primary storage having the *tag* doesn't have enough capacity	primaryStorage::allocator::userTag::SSD::required	0.6
primaryStorage::allocator::userTag::{tag}	if present, the VM's root volume will be allocated on the primary storage which has user tag *tag*, if there is any; NO failure will be raised if no primary storage has the *tag* or primary storage having the *tag* doesn't have enough capacity, instead, a random primary storage will be chosen.	primaryStorage::allocator::userTag::SSD	0.6

if more than one above system tags present on a disk offering, the precedent order is:

```
primaryStorage::allocator::uuid::{uuid} > primaryStorage::allocator::userTag::{tag}::required > prima
```

2.18 Virtual Machine

Table of contents

- *Virtual Machine*
  - *Overview*
  - *Inventory*
    - *Properties*
      - *Example*
      - *Location*
      - *Networks*
      - *VM Nic Inventory*
      - *Example*
      - *Volumes*
      - *Hypervisor Type*
      - *State*
  - *Operations*
    - *Create VM*
      - *Parameters*
      - *rootDiskOfferingUuid*
      - *dataDiskOfferingUuids*
    - *Stop VM*
      - *Parameters*
    - *Start VM*
      - *Parameters*
    - *Reboot VM*
      - *Parameters*
    - *Destroy VM*
      - *Parameters*
    - *Migrate VM*
      - *Parameters*
    - *Attach Data Volume*
    - *Detach Data volume*
    - *Query VM*
      - *Primitive Fields of Query*
      - *Nested And Expanded Fields of Query*
    - *Query VM Nic*
      - *Primitive Fields of Query Nic*
      - *Nested And Expanded Fields of Query Nic*
  - *Global Configurations*
    - *dataVolume.deleteOnVmDestroy*
  - *Tags*
    - *System Tags*
      - *HostName*

2.18.1 Overview

A virtual machine(VM) consumes datacenter resources of computing, storage, and network.

2.18.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
zoneUuid	uuid of ancestor zone, see *Zone* and *location*	true		0.6
clusterUuid	uuid of ancestor cluster, see *Cluster* and *location*	true		0.6
hostUuid	uuid of parent host the VM is currently running, see *Host* and *location*	true		0.6
lastHostUuid	uuid of parent host the VM was running last time, see *Host* and *location*	true		0.6
imageUuid	uuid of image from which the VM's root volume is created, see *Image*			0.6
instanceOfferingUuid	uuid of instance offering, see *Instance Offering*			0.6
rootVolumeUuid	uuid of VM's root volume, see *Volume*			0.6
defaultL3NetworkUuid	uuid of VM's default L3 network, see *L3 network* and *networks*			0.6
type	VM type • UserVm: created by users • ApplianceVm: created by ZStack to help manage the cloud		• UserVm • ApplianceVm	0.6
hypervisorType	VM's hypervisor type, see *Host* and *hypervisor type*		• KVM	0.6
state	VM's state, see *state*	• Created • Starting • Running • Stopping • Stopped • Rebooting • Destroying • Destroyed • Migrating • Unknown		0.6

Example


```
{
    "allVolumes": [
        {
            "createDate": "Dec 2, 2015 5:53:42 PM",
            "description": "Root volume for VM[uuid:d92a03ed745a0d32fe63dc30051d3862]",
            "deviceId": 0,
            "format": "qcow2",
            "installPath": "/opt/zstack/nfsprimarystorage/prim-a82b75ee064a48708960f42b800bd910/rootV
            "lastOpDate": "Dec 2, 2015 5:53:42 PM",
            "name": "ROOT-for-vm-4-vlan10",
            "primaryStorageUuid": "a82b75ee064a48708960f42b800bd910",
            "rootImageUuid": "f1205825ec405cd3f2d259730d47d1d8",
            "size": 419430400,
            "state": "Enabled",
            "status": "Ready",
            "type": "Root",
            "uuid": "e9555324042542288ec20a67797d476c",
            "vmInstanceUuid": "d92a03ed745a0d32fe63dc30051d3862"
        }
    ],
    "clusterUuid": "b429625fe2704a3e94d698ccc0fae4fb",
    "createDate": "Dec 2, 2015 5:53:42 PM",
    "defaultL3NetworkUuid": "6572ce44c3f6422d8063b0fb262cbc62",
    "hostUuid": "d07066c4de02404a948772e131139eb4",
    "hypervisorType": "KVM",
    "imageUuid": "f1205825ec405cd3f2d259730d47d1d8",
    "instanceOfferingUuid": "04b5419ca3134885be90a48e372d3895",
    "lastHostUuid": "d07066c4de02404a948772e131139eb4",
    "lastOpDate": "Dec 2, 2015 5:53:42 PM",
    "name": "vm-4-vlan10",
    "rootVolumeUuid": "e9555324042542288ec20a67797d476c",
    "state": "Running",
    "type": "UserVm",
    "uuid": "d92a03ed745a0d32fe63dc30051d3862",
    "vmNics": [
        {
            "createDate": "Dec 2, 2015 5:53:42 PM",
            "deviceId": 0,
            "gateway": "10.0.0.1",
            "ip": "10.0.0.218",
            "l3NetworkUuid": "6572ce44c3f6422d8063b0fb262cbc62",
            "lastOpDate": "Dec 2, 2015 5:53:42 PM",
            "mac": "fa:ef:34:5c:6c:00",
            "netmask": "255.255.255.0",
            "uuid": "fb8404455cf84111958239a9ec19ca28",
            "vmInstanceUuid": "d92a03ed745a0d32fe63dc30051d3862"
        }
    ],
    "zoneUuid": "3a3ed8916c5c4d93ae46f8363f080284"
}
```

Location

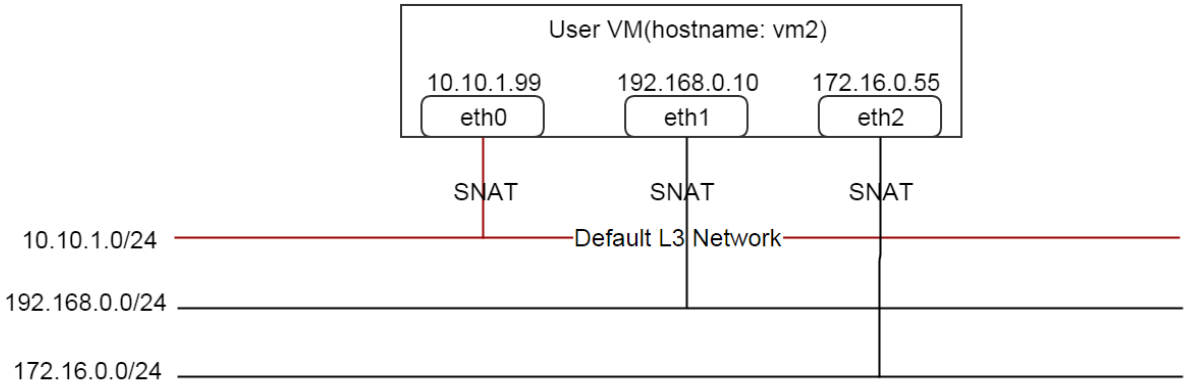
As ZStack arranges computing resources by zones, clusters, and hosts, a VM's location can be identified by zoneUuid, clusterUuid, and hostUuid. After a VM is running, those UUIDs will be set to values that represent the VM's current location; after stopped, the hostUuid is set to NULL, zoneUuid and clusterUuid are unchanged. The lastHostUuid is special, as it represents the host the VM run last time; for a new created VM, the lastHostUuid is NULL; once the VM is stopped, it's set to the previous value of the hostUuid.

The algorithms of selecting hosts for new created VMs are elaborated in *host allocator strategy*. In later of this chapter, strategies for starting VMs and migrating VMs will be demonstrated.

Networks

VMs can be on one or more *L3 networks*; *vm nics* encompass information like IP address, netmask, MAC of every L3 network. If a VM has more than one L3 networks, a default L3 network to provide default routing, DNS, and hostname must be specified; if a VM has only one L3 network, the one becomes the default L3 network automatically.

An example may help understand what is the default L3 network. Assuming you have a user vm like below picture:



The VM is on three L3 networks all providing SNAT service, and the default L3 network is 10.10.1.0/24:

```
CIDR: 10.10.1.0/24
Gateway: 10.10.1.1
DNS domain: web.tier.mycompany.com
```

then the VM's routing table is like:

```
default via 10.10.1.1 dev eth0
10.10.1.0/24 dev eth0  proto kernel  scope link  src 10.10.1.99
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.10
172.16.0.0/24 dev eth0  proto kernel  scope link  src 172.16.0.55
```

see the default routing is pointing to **10.10.1.1** that is the gateway of the default L3 network; and the VM's /etc/resolv.conf is like:

```
search web.tier.mycompany.com
nameserver 10.10.1.1
```

the DNS domain is from the default L3 network too; and the DNS name server is also the gateway **10.10.1.1** because the default L3 network provides the DNS server; at last, the FQDN(Full Qualified Domain Name) of the VM is like:

```
vm2.web.tier.mycompany.com
```

which is expanded by the DNS domain.

VM Nic Inventory

Name	Description	Optional	Choices	S
uuid	see *Resource Properties*			0
vmInstanceUuid	uuid of parent VM			0
l3NetworkUuid	uuid of *l3 network* the nic is on			0
ip	IP address			0
mac	MAC address			0
netmask	netmask			0
gateway	gateway			0
metaData	reserved field for internal use	true		0
deviceId	an integer that identifies nic's order in guest operating system's ethernet device list. For example, 0 usually means eth0, 1 usually means eth1.			0

In this ZStack version, once an IP is assigned to a VM nic, it will be with the nic through the entire life of the VM until the VM is destroyed.

Example

```
{
    "createDate": "Dec 2, 2015 5:53:42 PM",
    "deviceId": 0,
    "gateway": "10.0.0.1",
    "ip": "10.0.0.218",
    "l3NetworkUuid": "6572ce44c3f6422d8063b0fb262cbc62",
    "lastOpDate": "Dec 2, 2015 5:53:42 PM",
    "mac": "fa:ef:34:5c:6c:00",
    "netmask": "255.255.255.0",
    "uuid": "fb8404455cf84111958239a9ec19ca28",
    "vmInstanceUuid": "d92a03ed745a0d32fe63dc30051d3862"
}
```

Volumes

Field *allVolumes* is a list of *volume inventory* that contains the root volume and data volumes. To find out the root volume, users can iterate the list, either by checking if a volume's type is Root or using the field 'rootVolumeUuid' to match volumes' UUIDs. A root volume will be with the VM through its entire life until it's destroyed.

Hypervisor Type

VM's hypervisor type is inherited from image's hypervisor type or host's hypervisor type, depending on how the VM is created.

- **from a RootVolumeTemplate**:

  as the image already has operating system installed, the VM will be created on a host of the same hypervisor type to the image, so the VM's hypervisor type is inherited from the image.

- **from an ISO**: as the ISO will be used to install the VM's blank root volume, the VM can be created on hosts of any hypervisor types, then the VM's hypervisor type is inherited from the host it's created.

State

VMs have 10 states representing life cycles.

- **Created**

  The VM is just created as a record in database, but has not been started on any host. The state only exists when creating a new VM.

- **Starting**

  The VM is starting on a host

- **Running**

  The VM is running on a host

- **Stopping**

  The VM is stopping on a host

- **Stopped**

  The VM is stopped and not running on any host

- **Rebooting**

  The VM is rebooting on the host it's running previously

- **Destroying**

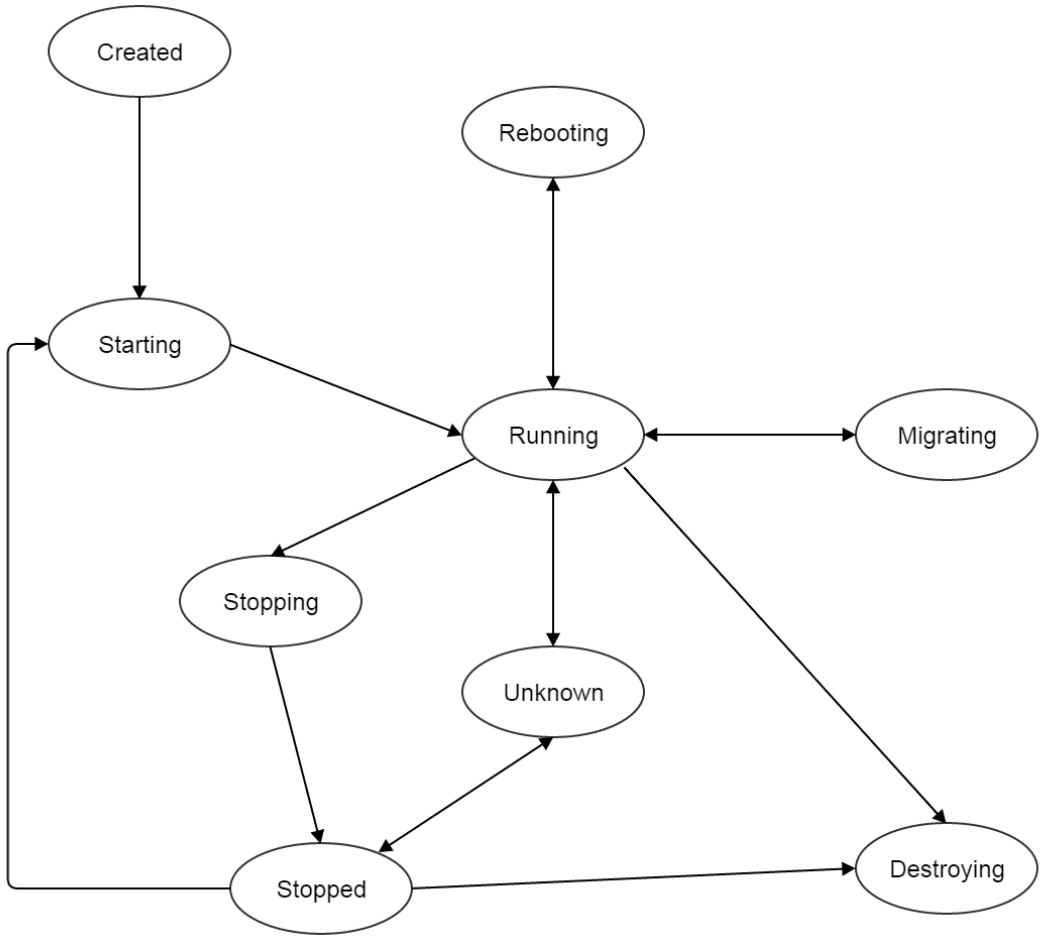
  The VM is being destroyed

- **Migrating**

  The VM is being migrated to another host

- **Unknown**

  For some reason, for example, losing connection to the host, ZStack fails to detect the VM's state



ZStack uses a VmTracer to periodically track VMs' states; the default interval is 60s. A VM's state may be changed outside ZStack, for example, a host power outage will make all VMs stop on the host; once the VmTracer detects a mismatch between the real state of a VM and the record in database, it will update database to catch up the real state. If the VmTracer fails to detect a VM's state, for example, because of losing connection between a ZStack management node and a host, it will place the VM into state Unknown; once the VmTracer successfully detects the VM's state again, for example, after connection recovers between the ZStack management node and the host, it will update the VM to the real state.

2.18.3 Operations

Create VM

Users can use CreateVmInstance to create a new VM. For example:

```
CreateVmInstance name=vm imageUuid=d720ff0c60ee48d3a2e6263dd3e12c33 instanceOfferingUuid=76789b62aeb5
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
instanceOfferingUuid	uuid of *instance offering*			0.6
imageUuid	uuid of *image*. Image can only be type of RootVolumeTemplate or ISO			0.6
l3NetworkUuids	a list of *L3 network* uuid			0.6
type	reserved field, default is UserVm		• UserVm • ApplianceVm	0.6
rootDiskOfferingUuid	uuid of *disk offering* for root volume, see *rootDiskOfferingUuid*	true		0.6
dataDiskOfferingUuids	a list of *disk offering* uuid, see *dataDiskOfferingUuids*	true		0.6
zoneUuid	if not null, the VM will be created in the specified zone; this field can be overridden by clusterUuid or hostUuid	true		0.6
clusterUuid	if not null, the VM will be created in the specified cluster; this field can be overridden by hostUuid	true		0.6
hostUuid	if not null, the VM will be created on the specified host	true		0.6
defaultL3NetworkUuid	if l3NetworkUuids includes more than one L3 network UUIDs, this field indicates which L3 network is the default L3 network. leave it alone if l3NetworkUuids has only one L3 network uuid.	true		0.6q

rootDiskOfferingUuid If a VM is created from an ISO image, users must specify a *disk offering* by rootDiskOfferingUuid so ZStack knows the disk size of the root volume; if the VM is created from an RootVolumeTemplate image, this field is ignored.

dataDiskOfferingUuids By providing a list of disk offering UUIDs in dataDiskOfferingUuids, users can create a VM with multiple data volumes attached. If a data volume failed to be created, the whole VM creation fails.

Stop VM

Users can use StopVmInstance to stop a running VM. For example:

```
StopVmInstance uuid=76789b62aeb542a5b4b8b8488fbaced2
```

Parameters

Name	Description	Optional	Choices	Since
uuid	VM uuid			0.6

Start VM

Users can use StartVmInstance to start a stopped VM. For example:

```
StartVmInstance uuid=76789b62aeb542a5b4b8b8488fbaced2
```

Parameters

Name	Description	Optional	Choices	Since
uuid	VM uuid			0.6

When starting a VM, ZStack uses LastHostPreferredAllocatorStrategy algorithm that will start the VM on the host it previously run if possible; otherwise, start the VM on a new host using the algorithm of *DesignatedHostAllocatorStrategy*.

Reboot VM

Users can use RebootVmInstance to reboot a running VM. For example:

```
RebootVmInstance uuid=76789b62aeb542a5b4b8b8488fbaced2
```

Parameters

Name	Description	Optional	Choices	Since
uuid	VM uuid			0.6

Destroy VM

Users can use DestroyVmInstance to destroy a VM. For example:

```
DestroyVmInstance uuid=76789b62aeb542a5b4b8b8488fbaced2
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	VM uuid			0.6

Warning: There is no way to recover a destroyed VM; once a VM is destroyed, its root volume will be deleted; if global configuration *dataVolume.deleteOnVmDestroy* is true, attached data volumes will be deleted as well; otherwise, data volumes will be detached.

Migrate VM

Admins can use MigrateVm to live migrate a running VM from the current host to another host. For example:

```
MigrateVm vmInstanceUuid=76789b62aeb542a5b4b8b8488fbaced2 hostUuid=37d3c4a1e2f14a1c8316a23531e62988
```

Parameters

Name	Description	Optional	Choices	Since
vmInstanceUuid	VM uuid			0.6
hostUuid	target host uuid; if omitted, ZStack will try to find a proper host automatically	true		0.6

A VM can migrate between two hosts only if their OS versions are exactly matching. For KVM, OS versions are determined by three system tags: *os::distribution*, *os::release*, and *os::version*.

When migrating, OS versions are checked by *MigrateVmAllocatorStrategy* which uses a similar algorithm of *DesignatedHostAllocatorStrategy* to choose target migration host.

Warning: For KVM, if you use customized libvirt and qemu rather than those builtin ones, migration may fail even OS versions match on two hosts. Please make sure OS version, libvirt version, and qemu version are all the same on two hosts for migration.

Attach Data Volume

See *attach volume to vm*.

Detach Data volume

See *detach volume from vm*.

Query VM

Users can use QueryVmInstance to query VMs. For example:

```
QueryVmInstance state=Running hostUuid=33107835aee84c449ac04c9622892dec
```

```
QueryVmInstance vmNics.eip.guestIp=10.23.109.23
```

Primitive Fields of Query

see *VM inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
vmNics	*VM nic inventory*	VM nics belonging to this VM	0.6
allVolumes	*volume inventory*	volumes belonging to this VM	0.6
zone	*zone inventory*	ancestor zone	0.6
cluster	*cluster inventory*	ancestor cluster	0.6
host	*host inventory*	parent host	0.6
image	*image inventory*	image this VM is created from	0.6
instanceOffering	*instance offering inventory*	instance offering this VM is created from	0.6
rootVolume	*volume inventory*	root volume belonging to this VM	0.6

Query VM Nic

Users can use QueryVmNic to query VM nics. For example:

```
QueryVmNic gateway=10.1.1.1
```

```
QueryVmNic eip.guestIp=11.168.2.13
```

Primitive Fields of Query Nic

see *VM nic inventory*

Nested And Expanded Fields of Query Nic

Field	Inventory	Since
vmInstance	*VM inventory*	0.6
l3Network	*L3 network inventory*	0.6
eip	*EIP inventory*	0.6
portForwarding	*port forwarding inventory*	0.6
securityGroup	*security group inventory*	0.6

2.18.4 Global Configurations

dataVolume.deleteOnVmDestroy

Name	Category	Default Value	Choices
dataVolume.deleteOnVmDestroy	vm	false	• true • false

If true, data volumes attached to the VM will be deleted as well when the VM is being deleted; otherwise, the data volumes will be detached.

2.18.5 Tags

Users can create user tags on a VM with resourceType=VmInstanceVO. For example:

```
CreateUserTag tag=web-server-vm resourceType=VmInstanceVO resourceUuid=a12b3cc9ee4440dfb00d41c1d2f72c
```

System Tags

HostName

Users can specify a hostname for a VM's default L3 network. This tag is usually specified in *systemTags* parameter when calling CreateVmInstance; if the default L3 network has a DNS domain, the hostname that VM's operating system receives will be automatically expanded with the DNS domain. For example, assuming the hostname is 'web-server' and DNS domain of the default L3 network is 'zstack.org', the final hostname will be 'web-server.zstack.org'.

Tag	Description	Example	Since
hostname::{hostname}	hostname for VM's default L3 network	hostname::web-server	0.6

For example:

```
CreateVmInstance name=vm systemTags=hostname::vm1 imageUuid=d720ff0c60ee48d3a2e6263dd3e12c33 instance
```

2.19 Security Group

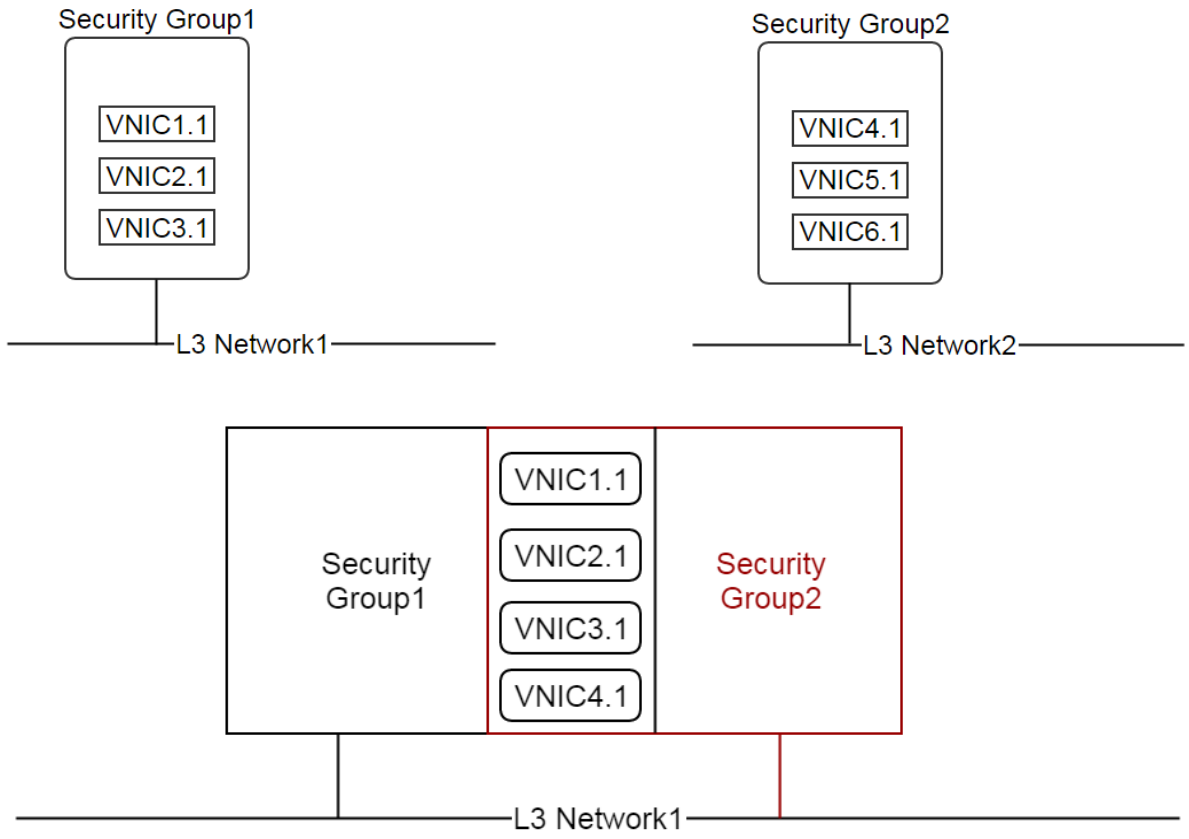
Table of contents

- Security Group
  - Overview
  - Security Group Inventory
    - Properties
    - Example
  - Security Group Rule Inventory
    - Properties
      - Traffic Type
      - Allowed CIDR
    - Example
  - Security Group And L3 Network
  - Operations
    - Create Security Group
      - Parameters
    - Add Rules To Security Group
      - Parameters
      - SecurityGroupRuleAO
    - Delete Rules From Security Group
      - Parameters
    - Add VM Nics Into Security Group
      - Parameters
    - Remove VM Nics from Security Group
      - Parameters
    - Attach Security Group To L3 Network
      - Parameters
    - Detach Security Group From L3 Network
      - Parameters
    - Delete Security Group
      - Parameters
    - Query Security Group
      - Primitive Fields
      - Nested And Expanded Fields
  - Global Configurations
    - ingress.defaultPolicy
    - egress.defaultPolicy
  - Tags

2.19.1 Overview

A security group acts as a virtual firewall that controls networking traffics of VMs. Depending on the isolation method a L2 network takes, users can use security group as firewalls or as a layer 3 isolation method. For example, if multiple tenants share a L3 network, every tenant can create a security group to protect their VMs from being accessed by other tenants. Tenants can also use security group along with *EIP* to control ports open to the public.

A security group consists of a set of rules that control ports' accessibility. A security group can be attached to one or more L3 networks; VM nics on attached L3 networks can join those security groups. A VM nic can join multiple security groups, rules applied to the nic will be merged.



The implementation of security group is hypervisor specific; not all hypervisors will support security group. In this ZStack version, security group is supported in KVM hypervisor by using IPTables.

Note: A large number of security group rules may hurt network performance because the hypervisor needs to check all rules against every network packet. ZStack will try to condense security group rules as much as possible; for example, if you specify two rules for two consecutive ports, they will be merged into one IPTable(for KVM) rule using a port range match.

To use security group, a L3 network must enable security group service using *AttachNetworkServiceToL3Network*, for example:

```
AttachNetworkServiceToL3Network l3NetworkUuid=50e637dc68b7480291ba87cbb81d94ad networkServices='{"1d
```

For VMs having multiple nics, all nics can join security groups.

The security group is essentially a distributed firewall; every rule change or nic join/leave event may lead firewall rules to be refreshed on multiple hosts. Given this fact, some security group APIs are implemented in an asynchronous manner, which may return before rules take effect on hosts. If there are more than one rules for a specific port, the most permissive rule takes effect. For example, if a rule1 allows traffic from 12.12.12.12 to access port 22 but a rule2 allows everyone to access port 22, the rule2 takes precedence.

2.19.2 Security Group Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
state	security group state; not implemented in this ZStack version		• Enabled • Disabled	0.6
rules	a list of *security group rule inventory*			0.6
attachedL3NetworkUuids	list of uuid of *L3 networks* that this security group has been attached			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

For an empty security group, there are default polices for ingress traffics and egress traffics; for ingress traffics, the default policy is to deny, which means all inbound traffics to the nics in this empty security group are blocked; for egress traffics, the default policy is to allow, which means all outbound traffics from the nics in this empty security group are allowed. To change default policies, admin can change global configuration *ingress.defaultPolicy* and *egress.defaultPolicy*.

Example


```
{
    "attachedL3NetworkUuids": [
        "0b48770e593e400c8f54e71fd4e7f514"
    ],
    "createDate": "Nov 16, 2015 1:02:22 AM",
    "lastOpDate": "Nov 16, 2015 1:02:22 AM",
    "name": "sg-in",
    "rules": [
        {
            "allowedCidr": "0.0.0.0/0",
            "createDate": "April 29, 2015 9:57:10 PM",
            "state": "Enabled",
            "endPort": 22,
            "lastOpDate": "Nov 29, 2015 9:57:10 PM",
            "protocol": "TCP",
            "securityGroupUuid": "9e0a72fe64814900baa22f78a1b9d235",
            "startPort": 22,
            "type": "Ingress",
            "uuid": "a338d11be18d4e288223597682964dc8"
        }
    ],
```

```
    "state": "Enabled",
    "uuid": "9e0a72fe64814900baa22f78a1b9d235"
}
```

2.19.3 Security Group Rule Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
securityGroupUuid	uuid of parent security group			0.6
type	see *traffic type*		• Ingress • Egress	0.6
protocol	traffic protocol type		• TCP • UDP • ICMP	0.6
startPort	when protocol is TCP/UDP, it's the start of port range; when protocol is ICMP, it's ICMP type		• for TCP/UDP: 0 - 65535 • for ICMP: see ICMP type and code , use '-1' to represent all types.	0.6
endPort	when protocol is TCP/UDP, it's the end of port range; when protocol is ICMP, it's ICMP code		• for TCP/UDP: 0 - 65535 • for ICMP: see ICMP type and code, use '-1' to represent all types.	0.6
allowedCidr	see *allowedCidr*			0.6
state	rule state, not implemented in this version		• Enabled • Disabled	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Traffic Type

There are two types of traffics:

- **Ingress**

  Inbound traffics that access a VM nic

- **Egress**

  Outbound traffics that leave from a VM nic

Allowed CIDR

Depending on traffic types, allowed CIDR has different meanings; its format is:

```
ipv4_address/network_prefix

for example: 12.12.12.12/24
```

if the traffic type is Ingress, allowed CIDR is a source CIDR that's allowed to reach VM nics; for example, a rule:

```
startPort: 22
endPort: 22
protocol: TCP
type: Ingress
allowedCidr: 12.12.12.12/32
```

means only TCP traffic **from** IP(12.12.12.12) is allowed to access port 22.

if the traffic type is Egress, allowed CIDR is a destination CIDR that's allowed to leave VM nics; for example, a rule:

```
startPort: 22
endPort: 22
protocol: TCP
type: Egress
allowedCidr: 12.12.12.12/32
```

means only TCP traffic **to** port 22 of IP 12.12.12.12 is allowed to leave.

The special CIDR 0.0.0.0/0 means all IP addresses.

Note: Allowed CIDR only controls IPs outside a security group. Rules are automatically applied to IPs of VM nics that are on the same L3 network and in the same security group. For example, if two nics: nic1(10.10.1.5) and nic2(10.10.1.6) are in the same security group which has a rule:

```
startPort: 22
endPort: 22
protocol: TCP
type: Ingress
allowedCidr: 12.12.12.12/32
```

nic1 and nic2 can reach port 22 of each other in spite of allowedCidr is set to 12.12.12.12/32.

Example

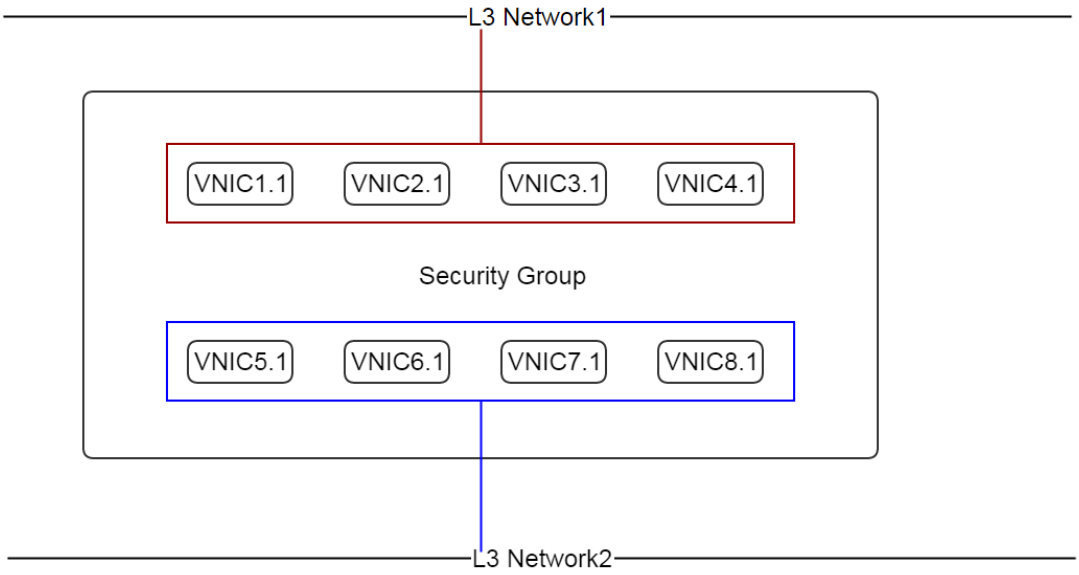
```
{
     "allowedCidr": "0.0.0.0/0",
     "state": "Enabled",
     "startPort": 22,
     "endPort": 22,
```

```
    "protocol": "TCP",
    "type": "Ingress",
    "createDate": "Nov 29, 2015 9:57:10 PM",
    "lastOpDate": "Nov 29, 2015 9:57:10 PM",
    "uuid": "a338d11be18d4e288223597682964dc8"
    "securityGroupUuid": "9e0a72fe64814900baa22f78a1b9d235",
}
```

2.19.4 Security Group And L3 Network

As having said, a security group can be attached to multiple L3 networks. The design consideration is that a security group is a set of firewall rules and can be applied to any L3 networks. For example, two different L3 networks may have the same set of firewall rules which make much sense to be put into the same security group.

VM nics from different L3 networks in the same security group are irrelevant. As mentioned in *Allowed CIDR*, VM nics of the same L3 network in a security group are not affected by rules' allowedCIDR, they can always reach ports opened of each other. However, if two nics in a security group are from different L3 networks, then the allowedCIDR will take effect when they try to reach each other.



If you find it's confusing to have a security group attached to multiple L3 networks, you can always create a security group per each L3 network.

2.19.5 Operations

Create Security Group

Users can use CreateSecurityGroup to create a security group. For example:

```
CreateSecurityGroup name=web
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6

Add Rules To Security Group

Users can use AddSecurityGroupRule to add rules to a security group. For example:

```
AddSecurityGroupRule securityGroupUuid=29a0f801f77b4b4f866fb4c9503d0fe9 rules="[{'type':'Ingress', 'p
```

This command executes asynchronously, it may return before rules are applied to all VM nics.

Parameters

Name	Description	Optional	Choices	Since
securityGroupUuid	uuid of security group			0.6
rules	a list of *SecurityGroupRuleAO*			0.6

SecurityGroupRuleAO

Name	Description	Optional	Choices	Since
type	traffic type, see *traffic type*		• Ingress • Egress	0.6
startPort	start port or ICMP type		• port: 0 - 65535 • ICMP type: see ICMP type and code	0.6
endPort	end port or ICMP code		• port: 0 - 65535 • ICMP code: see ICMP type and code	0.6
protocol	protocol type		• TCP • UDP • ICMP	0.6
allowedCidr	see *allowed CIDR*; default to 0.0.0.0/0	true		0.6

Delete Rules From Security Group

User can uses DeleteSecurityGroupRule to delete rules from a security group. For example:

```
DeleteSecurityGroupRule ruleUuids=a338d11be18d4e288223597682964dc8,9e0a72fe64814900baa22f78a1b9d235
```

This command executes asynchronously, it may return before rules are refreshed on all hosts.

Parameters

Name	Description	Optional	Choices	Since
ruleUuids	a list of uuid of *rule inventory*			0.6

Add VM Nics Into Security Group

Users can use AddVmNicToSecurityGroup to add VM nics to a security group. For example:

```
AddVmNicToSecurityGroup securityGroupUuid=0b48770e593e400c8f54e71fd4e7f514 vmNicUuids=b429625fe2704a
```

This command executes asynchronously, it may return before rules are applied on those nics.

Note: VM nics can only join security groups that have been attached to their L3 networks.

Parameters

Name	Description	Optional	Choices	Since
securityGroupUuid	security group uuid			0.6
vmNicUuids	a list of uuid of *vm nic inventory*			0.6

Remove VM Nics from Security Group

Users can use DeleteVmNicFromSecurityGroup to delete VM nics from a security group. For example:

```
DeleteVmNicFromSecurityGroup securityGroupUuid=0b48770e593e400c8f54e71fd4e7f514 vmNicUuids=b429625fe
```

This command executes asynchronously, it may return before rules are refreshed on nics in the security group.

Parameters

Name	Description	Optional	Choices	Since
securityGroupUuid	security group uuid			0.6
vmNicUuids	a list of uuid of *vm nic inventory*			0.6

Attach Security Group To L3 Network

Users can use AttachSecurityGroupToL3Network to attach a security group to a L3 network. For example:

```
AttachSecurityGroupToL3Network securityGroupUuid=0b48770e593e400c8f54e71fd4e7f514 l3NetworkUuid=95de
```

Note: A security group can only be attached to L3 networks that have security group network service enabled

Parameters

Name	Description	Optional	Choices	Since
securityGroupUuid	security group uuid			0.6
l3NetworkUuid	L3 network uuid			0.6

Detach Security Group From L3 Network

Users can use DetachSecurityGroupFromL3Network to detach a security group from a L3 network:

```
DetachSecurityGroupFromL3Network securityGroupUuid=0b48770e593e400c8f54e71fd4e7f514 l3NetworkUuid=95c
```

After detaching, all rules will be removed from VM nics of the L3 network and in this security group. This command executes asynchronously, it may return before rules are refreshed on those nics.

Parameters

Name	Description	Optional	Choices	Since
securityGroupUuid	security group uuid			0.6
l3NetworkUuid	L3 network uuid			0.6

Delete Security Group

Users can use DeleteSecurityGroup to delete a security group. For example:

```
DeleteSecurityGroup uuid=0b48770e593e400c8f54e71fd4e7f514
```

After deleting, all rules will be removed from VM nics in this security group. This command executes asynchronously, it may return before rules are refreshed on those VM nics.

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	security group uuid			0.6

Query Security Group

Users can use QuerySecurityGroup to query security groups. For example:

```
QuerySecurityGroup rules.startPort=22 rules.type=Ingress rules.protocol=TCP
```

```
QuerySecurityGroup vmNic.ip=192.168.0.205
```

Primitive Fields

see *security group inventory*.

Nested And Expanded Fields

Field	Inventory	Description	Since
rules	*security group rule inventory*	rules the security group has	0.6
vmNic	*VM nic inventory*	VM nics that have joined this security group	0.6
l3Network	*L3 network inventory*	L3 networks this security group is attached	0.6

2.19.6 Global Configurations

ingress.defaultPolicy

Name	Category	Default Value	Choices
ingress.defaultPolicy	securityGroup	deny	• deny • accept

The default ingress policy for empty security groups.

egress.defaultPolicy

Name	Category	Default Value	Choices
egress.defaultPolicy	securityGroup	accept	• deny • accept

The default egress policy for empty security groups.

2.19.7 Tags

Users can create user tags on a security group with resourceType=SecurityGroupVO. For example:

```
CreateUserTag tag=web-tier-security-group resourceType=SecurityGroupVO resourceUuid=f25a28fdb21147f8
```

2.20 Network Services And Virtual Router

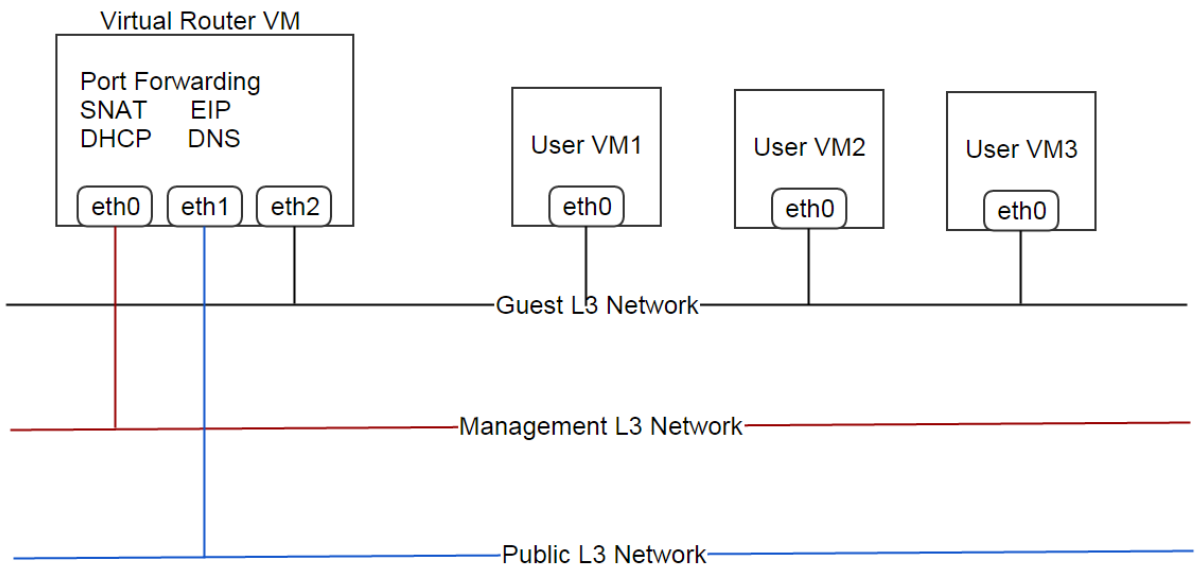
Table of contents

- Network Services And Virtual Router
  - Overview
  - Network typology
  - Virtual Router Network Services
  - Inventory
    - Properties
    - Example
  - Virtual Router Offering
    - Inventory
      - Properties
      - Example
      - Default Offering
      - Image
      - Management Network and Public Network
  - Operations
    - Create Virtual Router Offering
      - Parameters
    - Delete Virtual Router Offering
    - Reconnect Virtual Router Agent
      - Parameters
    - Start Virtual Router VM
    - Reboot Virtual Router VM
    - Stop Virtual Router VM
    - Destroy Virtual Router VM
    - Migrate Virtual Router VM
    - Create Virtual Router VM
    - Query Virtual Router VM
      - Primitive Fields
      - Nested And Expanded Fields
    - Query Virtual Router Offering
      - Primitive Fields
      - Nested And Expanded Fields
  - Global Configurations
    - agent.deployOnStart
    - command.parallelismDegree
    - connect.timeout
    - agent.deployOnStart
  - Tags
    - System Tags
      - Parallel Command Level
      - Guest L3 Network

2.20.1 Overview

ZStack supports a couple of OSI layer 4 ~ 7 network services: DHCP, DNS, SNAT, EIP, and PortForwarding. A L3 network can enable network services supplied by providers attached to its parent L2 network. Check *Network Services* for a list of supported network services.

ZStack comes with a builtin network service provider – Virtual Router Provider, which uses customized Linux VMs to implement network services. When creating a new VM on a L3 network that has network services attached from the virtual router provider, a virtual router VM known as appliance VM will be created if there isn't one yet.



Computing capacity(CPU, Memory) of a virtual router VM is defined by a special instance offering called virtual router offering. Besides CPU and memory, several extra parameters like image, management L3 network, public L3 network can be defined in a virtual router offering; details can be found in *virtual router offering inventory*.

Though the virtual router provider is the only network service provider(except security group provider) in current ZStack version, the network services framework is highly pluggable that vendors can easily add their implementation by implementing small plugins.

2.20.2 Network typology

A virtual router VM typically has three L3 networks:

- **Management Network**:

  The network that ZStack management nodes communicate to virtual router agents; eth0 is the nic on the management network.

- **Public Network**:

  The network that provides internet access, and provides public IPs for user VMs that use EIP, port forwarding, and source NAT; eth1 is the nic on the public network.

  Note: A RFC 1918 private subnet can be used as a public network as long as it can reach internet.

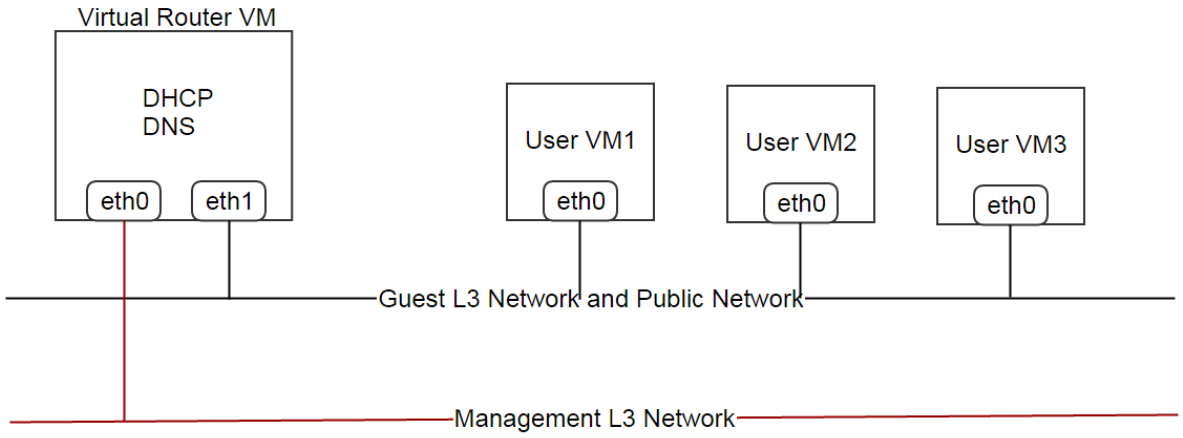
- **Guest Network**

  The network where user VMs connect. eth2 is the nic on the guest network.

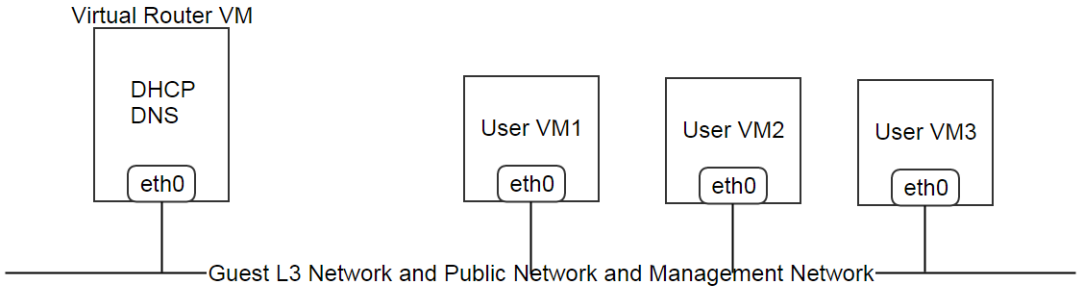
In a normal setup, all three networks should be separate L3 networks; however, two or even three networks can be combined to one network, depending on what network typology you want.

For a *flat network*, a virtual router VM provides only DHCP and DNS services, the network typologies can be:

- **Combined public network and guest network; a separate management network**

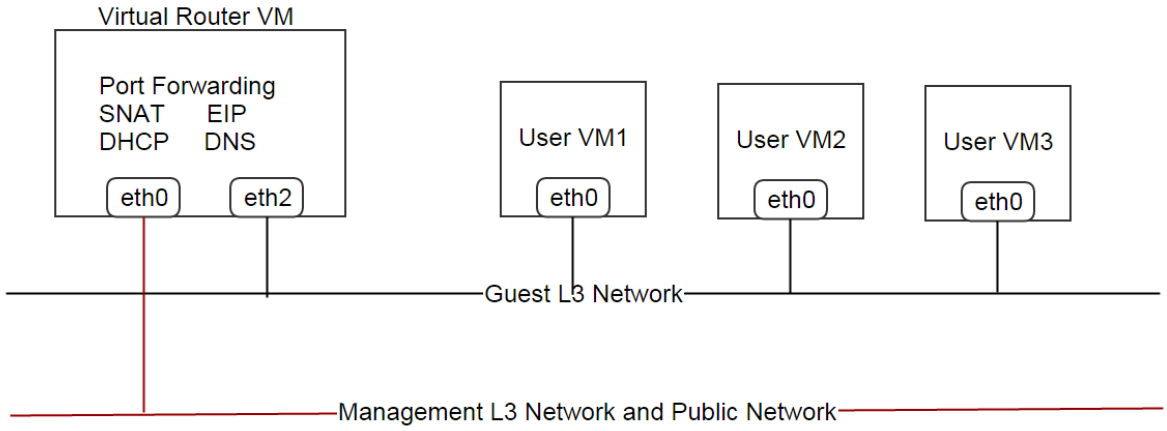


- **Combined all of public network, guest network, and management network**

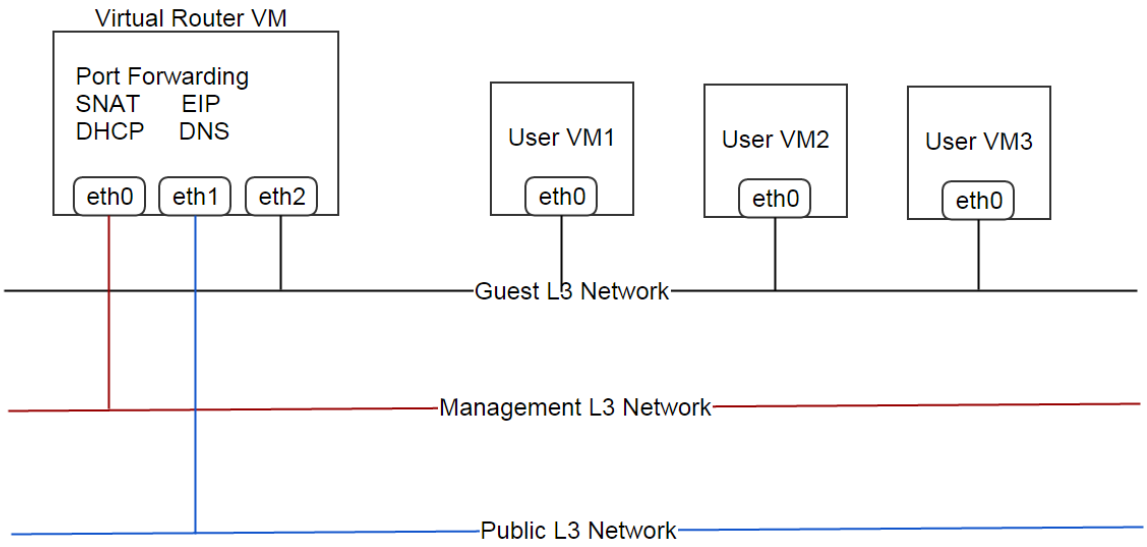


For a *private network or isolated network*, a virtual router VM provides DHCP, DNS, SNAT; and may provide EIP and Port Forwarding too, depending on users' choices; the network typologies can be:

- **Combined public network and management network; a separate guest network**



- **Separate public network, management network, and guest network**



Note: Because SSH port 22 is open on the management network, combining management network with other networks may lead to security issues. It's highly recommended to use a separate management network.

Note: VPC is not supported in this ZStack version.

2.20.3 Virtual Router Network Services

In this ZStack version, the virtual router provider provides five network services: DHCP, DNS, SNAT, EIP, and Port-Forwarding; we will talk about EIP and Port Forwarding in dedicated chapters because they have own APIs.

- **DHCP**

  The virtual router VM acts as a DHCP server on the guest L3 network; the virtual router DHCP server uses static IP-MAC mapping so user VMs always get the same IP address.

- **DNS**

  The virtual router VM, no matter the DNS service is enabled or not, is always the DNS server of the guest L3 network. If the DNS service is enabled, DNS of the guest L3 network will be set as upstream DNS servers of the virtual router VM. See *L3 network* for how to add DNS to a L3 network.

- **SNAT**

  The virtual router VM acts as a router and provides source NAT to user VMs.

2.20.4 Inventory

Besides properties included in the *VM instance inventory*, the virtual router VM has some extra properties.

Properties

Name	Description	Optional	Choices	Since
applianceVmType	appliance VM type		• VirtualRouter	0.6
managementNetworkUuid	management L3 network uuid			0.6
defaultRouteL3NetworkUuid	the uuid of L3 network which provides default routing in the virtual router VM			0.6
publicNetworkUuid	the public L3 network uuid			0.6
status	virtual router agent status		• Connecting • Connected • Disconnected	0.6

Example


```
{
    "allVolumes": [
        {
            "createDate": "August 2, 2015 5:54:12 PM",
            "description": "Root volume for VM[uuid:f1e76cb2ef0c4dfa87f3b807eb4d7437]",
            "deviceId": 0,
            "format": "qcow2",
            "installPath": "/opt/zstack/nfsprimarystorage/prim-a82b75ee064a48708960f42b800bd910/rootV
            "lastOpDate": "Dec 2, 2015 5:54:12 PM",
            "name": "ROOT-for-virtualRouter.l3.8db7eb2ccdab4c4eb4784e46895bb016",
            "primaryStorageUuid": "a82b75ee064a48708960f42b800bd910",
            "rootImageUuid": "b4fe2ebbc4522e199d36985012254d7d",
            "size": 462945280,
            "state": "Enabled",
            "status": "Ready",
            "type": "Root",
            "uuid": "2acccd875e364b53824def6248c94a51",
            "vmInstanceUuid": "f1e76cb2ef0c4dfa87f3b807eb4d7437"
        }
    ],
    "applianceVmType": "VirtualRouter",
    "clusterUuid": "b429625fe2704a3e94d698ccc0fae4fb",
    "createDate": "Dec 2, 2015 5:54:12 PM",
    "defaultRouteL3NetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
    "hostUuid": "d07066c4de02404a948772e131139eb4",
    "hypervisorType": "KVM",
    "imageUuid": "b4fe2ebbc4522e199d36985012254d7d",
    "instanceOfferingUuid": "f50a232a1448401cb8d049aad9c3860b",
    "lastHostUuid": "d07066c4de02404a948772e131139eb4",
    "lastOpDate": "Dec 2, 2015 5:54:12 PM",
    "managementNetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
    "name": "virtualRouter.l3.8db7eb2ccdab4c4eb4784e46895bb016",
    "rootVolumeUuid": "2acccd875e364b53824def6248c94a51",
    "publicNetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
```

```
    "state": "Running",
    "status": "Connected",
    "type": "ApplianceVm",
    "uuid": "f1e76cb2ef0c4dfa87f3b807eb4d7437",
    "vmNics": [
        {
            "createDate": "Dec 2, 2015 5:54:12 PM",
            "deviceId": 1,
            "gateway": "10.1.1.1",
            "ip": "10.1.1.155",
            "l3NetworkUuid": "8db7eb2ccdab4c4eb4784e46895bb016",
            "lastOpDate": "Dec 2, 2015 5:54:12 PM",
            "mac": "fa:99:e7:31:98:01",
            "metaData": "4",
            "netmask": "255.255.255.0",
            "uuid": "30bd463b926e4299a1326293ee75ae13",
            "vmInstanceUuid": "f1e76cb2ef0c4dfa87f3b807eb4d7437"
        },
        {
            "createDate": "Dec 2, 2015 5:54:12 PM",
            "deviceId": 0,
            "gateway": "192.168.0.1",
            "ip": "192.168.0.188",
            "l3NetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
            "lastOpDate": "Dec 2, 2015 5:54:12 PM",
            "mac": "fa:74:3f:40:cb:00",
            "metaData": "3",
            "netmask": "255.255.255.0",
            "uuid": "dc02fee25e9244ad8cbac151657a7b34",
            "vmInstanceUuid": "f1e76cb2ef0c4dfa87f3b807eb4d7437"
        }
    ],
    "zoneUuid": "3a3ed8916c5c4d93ae46f8363f080284"
}
```


2.20.5 Virtual Router Offering

A virtual router offering is an *instance offering* with some extra properties.

Inventory

Besides properties in *instance offering inventory*, the virtual router offering has below additional properties:

Properties

managementNetworkUuid	management L3 network uuid			0.6
publicNetworkUuid	public L3 network uuid			0.6
zoneUuid	uuid of ancestor zone. A virtual router VM will only be created from a virtual router offering in the same zone.			0.6
isDefault	see :*default offering*			0.6
imageUuid	virtual router image uuid, see *image*			0.6

Example


```
{
    "allocatorStrategy": "DefaultHostAllocatorStrategy",
    "cpuNum": 1,
    "cpuSpeed": 128,
    "createDate": "Nov 30, 2015 3:31:43 PM",
    "imageUuid": "b4fe2ebbc4522e199d36985012254d7d",
    "isDefault": true,
    "lastOpDate": "Nov 30, 2015 3:31:43 PM",
    "managementNetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
    "memorySize": 536870912,
    "name": "VROFFERING5",
    "publicNetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
    "sortKey": 0,
    "state": "Enabled",
    "type": "VirtualRouter",
    "uuid": "f50a232a1448401cb8d049aad9c3860b",
    "zoneUuid": "3a3ed8916c5c4d93ae46f8363f080284"
}
```


Default Offering When creating a virtual router VM on a L3 network, ZStack needs to decide what virtual router offering to use; the strategy is:

1. use a virtual router offering if it has a system tag *guestL3Network* that includes the L3 network's uuid.
2. use the default virtual router offering if nothing found in step 1.

for every zone, there must be a default virtual router offering.

Image A virtual router VM uses a customized Linux image that can be download from http://download.zstack.org/templates/zstack-virtualrouter-0.6.qcow2. The root credential of the Linux operating system is:

```
username: root
password: password
```

users who have console access to the virtual router VM can use this credential to login.

Before creating a virtual router offering, users need to add the image to a backup storage using command *add image*; to prevent creating user VMs from this image, users can set parameter 'system' to true.

Note: In future ZStack version, there will be a feature that generates random passwords for the root account, which makes the virtual router VM more secure.

Management Network and Public Network Before creating a virtual router offering, users must create those L3 networks using command *create L3 network*. To prevent creating user VMs on those networks, users can set parameter 'system' to true.

2.20.6 Operations

Create Virtual Router Offering

Users can use CreateVirtualRouterOffering to create a virtual router offering. For example:

```
CreateVirtualRouterOffering name=small cpuNum=1 cpuSpeed=1000 memorySize=1073741824 isDefault=true
managementNetworkUuid=95dede673ddf41119cbd04bcb5d73660 publicNetworkUuid=8db7eb2ccdab4c4eb4784e46895
imageUuid=95dede673ddf41119cbd04bcb5d73660
```

Besides parameters that *CreateInstanceOffering* has, there are additional parameters:

Parameters

Name	Description	Optional	Choices	Since
managementNetworkUuid	uuid of management L3 network			0.6
publicNetworkUuid	uuid of public L3 network; default to managementNetworkUuid.	true		0.6
zoneUuid	uuid of ancestor zone			0.6
imageUuid	image uuid			0.6

Delete Virtual Router Offering

see *DeleteInstanceOffering*

Reconnect Virtual Router Agent

As mentioned before, there is a Python virtual router agent inside the virtual router VM. Users can use ReconnectVirtualRouter to reinitialize a connection process from a ZStack management node to a virtual router VM, which will:

1. Upgrade the virtual router agent if the md5sum of the agent binary doesn't match the md5sum of the one in the management node's agent repository.
2. Restart the agent
3. Reapply all network services configurations including DHCP, DNS, SNAT, EIP, and PortForwarding to the virtual router VM.

A command example is like:

```
ReconnectVirtualRouter vmInstanceUuid=bd1652b1e44144e6b9b5b286b82edb69
```

Parameters

Name	Description	Optional	Choices	Since
vmInstanceUuid	virtual router VM uuid			0.6

Start Virtual Router VM

see *StartVmInstance*. While starting, the virtual router VM will perform agent connection process described in *ReconnectVirtualRouter*.

Reboot Virtual Router VM

see *RebootVmInstance*. While rebooting, the virtual router VM will perform agent connection process described in *ReconnectVirtualRouter*.

Stop Virtual Router VM

see *StopVmInstance*.

Warning: After the virtual router VM stops, user VMs on the guest L3 network served by the virtual router VM may lose their network functions.

Destroy Virtual Router VM

see *DestroyVmInstance*.

Warning: After the virtual router VM is destroyed, user VMs on the guest L3 network served by the virtual router VM may lose their network functions.

Migrate Virtual Router VM

see *MigrateVm*.

Create Virtual Router VM

Though there is no ready API to create a virtual router VM manually, users can trigger an automatic creation by creating or staring a user VM on the guest L3 network. If the L3 network doesn't have a virtual router VM running, creating, or stopping/starting a user VM will trigger the creation of a virtual router VM.

Query Virtual Router VM

Users can use QueryVirtualRouterVm to query virtual router VMs. For example:

```
QueryVirtualRouterVm defaultRouteL3NetworkUuid=95dede673ddf41119cbd04bcb5d73660
```

```
QueryVirtualRouterVm vmNics.mac=fa:d9:af:a1:38:01
```

Primitive Fields

see *appliance vm inventory*.

Nested And Expanded Fields

Field	Inventory	Description	Since
vmNics	*VM nic inventory*	VM nics of the virtual router VM	0.6
allVolumes	*volume inventory*	volumes of the virtual router VM	0.6
host	*host inventory*	host the virtual router VM is running	0.6
cluster	*cluster inventory*	cluster the virtual router VM belongs	0.6
image	*image inventory*	image from which the virtual router VM is created	0.6
zone	*zone inventory*	zone the virtual router VM belongs	0.6
rootVolume	*volume inventory*	root volume of the virtual router VM	0.6
virtualRouterOffering	*virtual router offering inventory*		0.6
eip	*EIP inventory*	EIP that the virtual router VM serves	0.6
vip	*VIP inventory*	VIP that the virtual router VM serves	0.6
portForwarding	*port forwarding rule inventory*	port forwarding rule that the virtual router VM serves	0.6

Query Virtual Router Offering

Users can use QueryVirtualRouterOffering to query virtual router offerings. For example:

```
QueryVirtualRouterOffering managementNetworkUuid=a82b75ee064a48708960f42b800bd910 imageUuid=6572ce44
```

```
QueryVirtualRouterOffering managementL3Network.name=systemL3Network image.name=newVirtualRouterImage
```

Primitive Fields

see *virtual router offering inventory*.

Nested And Expanded Fields

Field	Inventory	Description	Since
image	*image inventory*	image the offering contains	0.6
managementL3Network	*L3 network inventory*	management L3 network the offering contains	0.6
publicL3Network	*L3 network inventory*	public L3 network the offering contains	0.6
zone	*zone inventory*	zone the offering belongs to	0.6

2.20.7 Global Configurations

agent.deployOnStart

Name	Category	Default Value	Choices
agent.deployOnStart	virtualRouter	false	• true • false

Whether to deploy a virtual router agent when a virtual router VM starts/stops/reboots; as the virtual router agent is builtin in the virtual router VM, this value should only be set to true when users want to upgrade the agent.

command.parallelismDegree

Name	Category	Default Value	Choices
command.parallelismDegree	virtualRouter	100	> 0

The max number of concurrent commands that can be executed by the virtual router agent.

connect.timeout

Name	Category	Default Value	Choices
connect.timeout	applianceVm	300	> 0

The connecting timeout of SSH connection when management nodes connect virtual router agents, in seconds. If a management node cannot establish a SSH connection to a virtual router VM within the given timeout, an error will be raised.

agent.deployOnStart

Name	Category	Default Value	Choices
agent.deployOnStart	applianceVm	false	• true • false

Whether to deploy an appliance VM agent when an appliance VM starts/stops/reboots; as the agent is builtin in the appliance VM, this value should only be set to true when users need to upgrade the agent.

Note: There are actually two agents in virtual router VM, one is virtual router agent and another is appliance VM agent. They work for different purposes, users normally don't need to care about them.

2.20.8 Tags

Users can create user tags on a virtual router offering or a virtual router VM using the same way mentioned in chapter of instance offering and chapter of virtual macine.

System Tags

Parallel Command Level

Admins can limit max number of commands that can be executed in parallel in a virtual router VM.

Tag	Description	Example	Since
commandsParallelismDegree::{parallelismDegree}	the max number of commands that can be executed in parallel in a virtual router VM	commandsParallelismDegree::100	0.6

This tag can be created on a virtual router offering or a virtual router VM; if it's on a virtual router offering, virtual router VMs created form the offering will inherit the tag. Please use resourceType=InstanceOfferingVO for virtual router offerings, resourceType=VmInstanceVO for virtual router VMs.

Guest L3 Network

Admins can bind a virtual router offering to a guest L3 network, in order to specify which virtual router offering to use when creating a virtual router VM on the guest L3 network.

Tag	Description	Example	Since
guestL3Network::{guestL3NetworkUuid}	the uuid of guest L3 network	guestL3Network::dd56c5c209a74b669b3fe6116a611d57	3.6

For example:

```
CreateSystemTag resourceType=InstanceOfferingVO resourceUuid=YOUR_VR_OFFERING_UUID tag=guestL3Networ
```

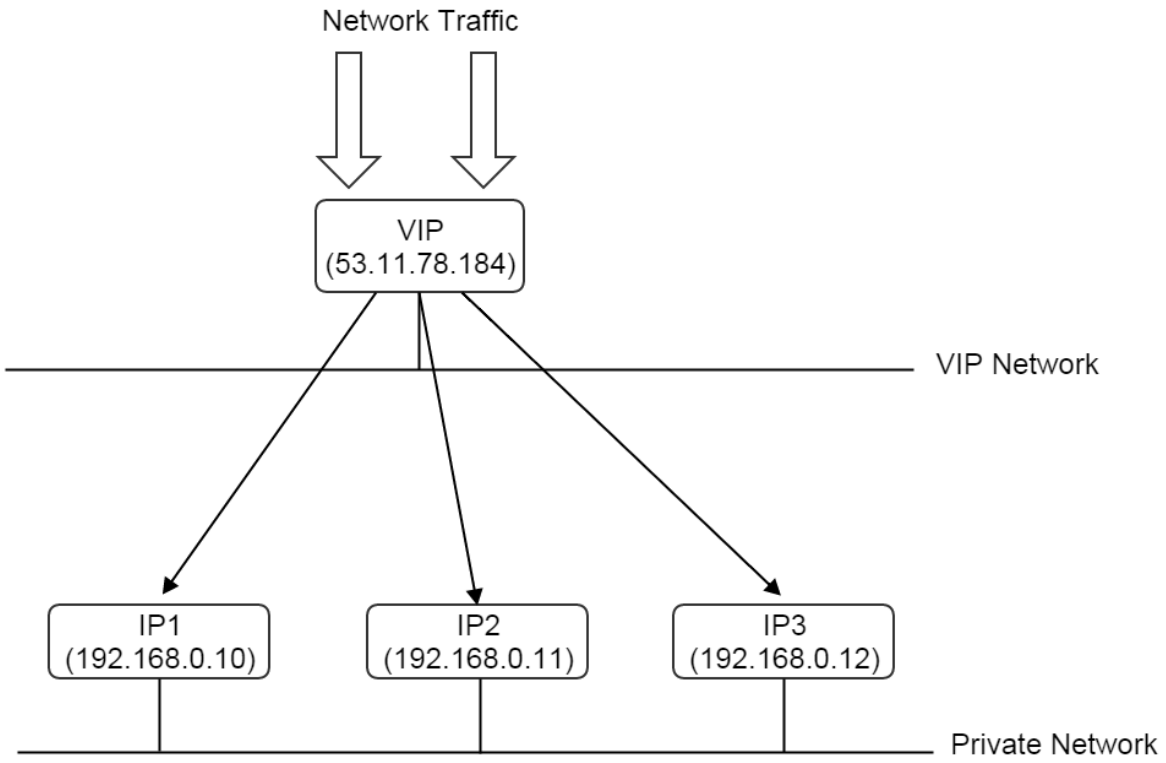
2.21 Virtual IP Address

Table of contents

- *Virtual IP Address*
  - *Overview*
  - *Inventory*
    - *Properties*
    - *Example*
  - *Operations*
    - *Create VIP*
      - *Parameters*
      - *RequiredIp*
    - *Delete VIP*
    - *Query VIP*
      - *Primitive Fields*
      - *Nested And Expanded Fields*
  - *Tags*

2.21.1 Overview

When bridging communication between two networks, many network services such as Port Forwarding, EIP, VPN, Load Balancing need virtual Ip addresses (VIP); incoming packets are sent to VIPs and are routed to private network IPs.



In real world cases, VIPs are usually public IPs that can be reached by the internet, routing traffics to behind private IPs which are often on a private network not visible to the internet.

In this ZStack version, a VIP must be allocated before creating a port forwarding rule or an EIP. For this time being, as the virtual router provider is the only network service provider, a VIP should be created from a virtual router VM's public network(see *virtual router offering*) in order to route traffics to the guest network.

2.21.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
ipRangeUuid	uuid of IP range the VIP is allocated			0.6
l3NetworkUuid	uuid of L3 network the VIP is allocated			0.6
ip	IP address			0.6
state	VIP state, not implemented in this version		• Enabled • Disabled	0.6
gateway	gateway			0.6
netmask	netmask			0.6
serviceProvider	name of service provider that uses this VIP	true		0.6
peerL3NetworkUuid	uuid of L3 network to which this VIP routes traffic			0.6
useFor	the service name which uses the VIP	true	• EIP • PortForwarding	0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example


```
{
    "createDate": "Nov 28, 2015 6:52:01 PM",
    "gateway": "192.168.0.1",
    "ip": "192.168.0.189",
    "l3NetworkUuid": "95dede673ddf41119cbd04bcb5d73660",
    "lastOpDate": "Nov 28, 2015 6:52:01 PM",
    "name": "vip-905d8a5c191c6e30173037e9d4c0ec56",
    "netmask": "255.255.255.0",
    "peerL3NetworkUuid": "6572ce44c3f6422d8063b0fb262cbc62",
    "serviceProvider": "VirtualRouter",
    "state": "Enabled",
    "useFor": "Eip",
    "uuid": "429106d5a63a4995911c2c5f14299b85"
}
```

2.21.3 Operations

Create VIP

Users can use CreateVip to create a VIP. For example:

```
CreateVip name=vip1 l3NetworkUuid=95dede673ddf41119cbd04bcb5d73660
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
l3NetworkUuid	uuid of the L3 network that the VIP will be allocated			0.6
requiredIp	the IP address you want to acquire, see *requiredIp*			0.6
allocatorStrategy	the algorithm of allocating a VIP		• RandomIpAllocatorStrategy	0.6

RequiredIp Users can instruct ZStack to allocate a specific VIP by specifying 'requiredIp', as long as the IP is still available on the target L3 network.

Delete VIP

Users can use DeleteVip to delete a VIP. For example:

```
DeleteVip uuid=429106d5a63a4995911c2c5f14299b85
```

> **Warning:** If there is a network service bound to the VIP, for example, an EIP; the network service entity(an EIP or a port forwarding rule) will be deleted automatically as well.

Query VIP

Users can use QueryVip to query a VIP. For example:

```
QueryVip ip=17.16.89.2 serviceProvider!=null
```

```
QueryVip eip.guestIp=10.256.99.2
```

Primitive Fields

see *VIP inventory*

Nested And Expanded Fields

Field	Inventory	Description	Since
eip	*EIP inventory*	the EIP that the VIP is bound to	0.6
portForwarding	port forwarding rule inventory	the port forwarding rule that the VIP is bound to	0.6

2.21.4 Tags

Users can create user tags on a VIP with resourceType=VipVO. For example:

```
CreateUserTag tag=web-tier-vip resourceType=VipVO resourceUuid=c3206d0e29074e21984c584074c63920
```

2.22 Elastic Port Forwarding

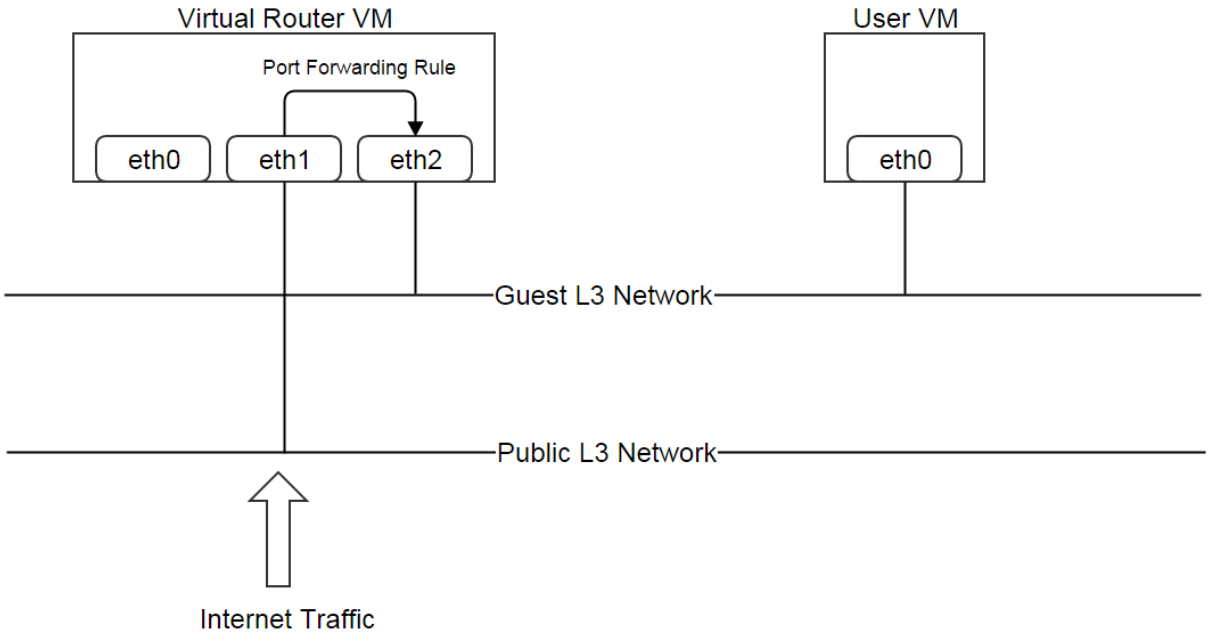
Table of contents

- *Elastic Port Forwarding*
  - *Overview*
  - *Port Forwarding Rule Inventory*
    - *Properties*
    - *Example*
  - *Operations*
    - *Create Port Forwarding Rule*
      - *Parameters*
    - *Delete Port Forwarding Rule*
      - *Parameters*
    - *Attach Port Forwarding Rule*
      - *Parameters*
    - *Detach Port Forwarding Rule*
      - *Parameters*
    - *Query Port Forwarding Rule*
      - *Primitive Fields*
      - *Nested And Expanded Fields*
  - *Global Configurations*
    - *snatInboundTraffic*
  - *Tags*

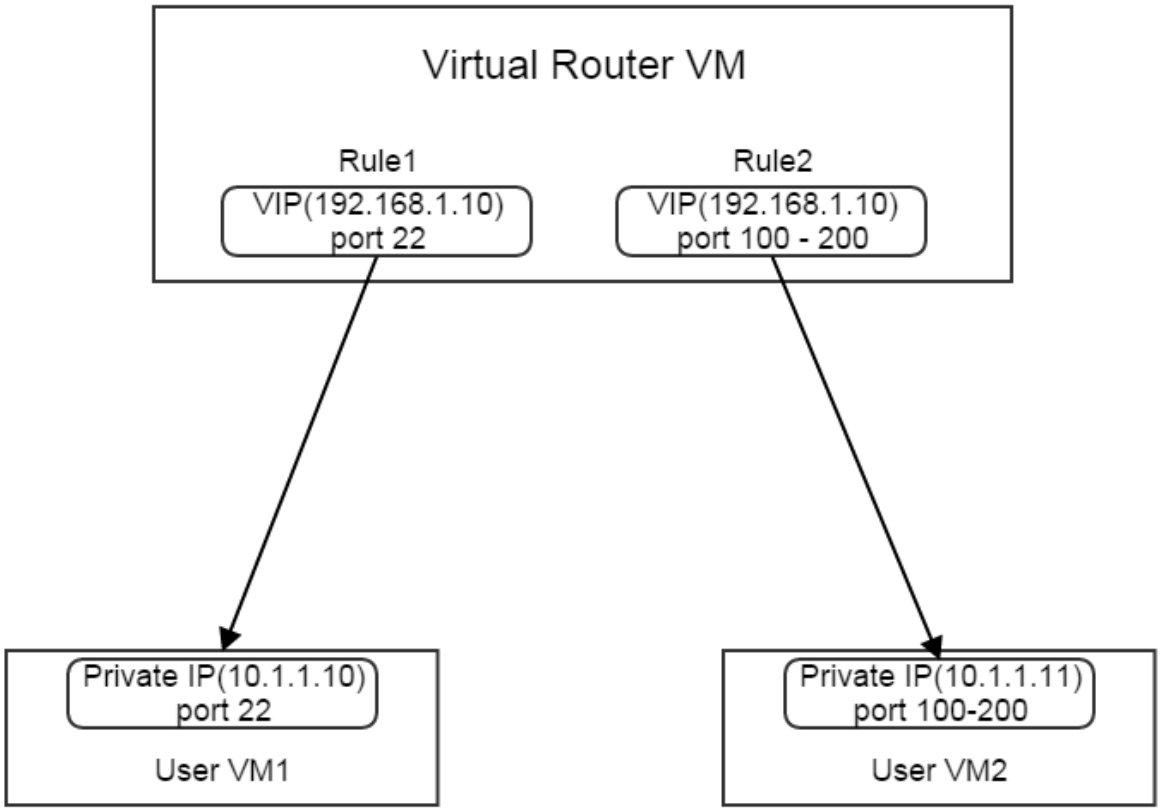
2.22.1 Overview

When user VMs are on a *private network or isolated network* with SNAT service enabled, they can reach outside network but cannot be reached by outside network, which is the nature of SNAT. Users can create port forwarding rules to allow outside network to reach specific ports of user VMs behind SNAT. ZStack supports elastic port forwarding rules, which means rules can be attached/detached to/from VMs on demand.

As the virtual router provider is the only network service provider in this ZStack version, a port forwarding rule is actually created between a virtual router VM's public network and guest network.



A VIP can be used for multiple port forwarding rules, as long as rules' port ranges don't overlap; for example:



2.22.2 Port Forwarding Rule Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
vipIp	IP address of VIP			0.6
guestIp	IP address of VM nic	true		0.6
vipUuid	uuid of VIP			0.6
vipPortStart	the start port of VIP		1 ~ 65535	0.6
vipPortEnd	the end port of VIP		1 ~ 65535	0.6
privatePortStart	the start port of guest IP		1 ~ 65535	0.6
privatePortEnd	the end port of guest IP		1 ~ 65535	0.6
vmNicUuid	uuid of guest VM nic	true		0.6
protocolType	protocol type of network traffic		• TCP • UDP	0.6
state	rule state, not implemented in this version		• Enabled • Disabled	0.6
allowedCidr	source CIDR; the port forwarding rule only applies to traffics with this source CIDR			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example


```
{
    "allowedCidr": "0.0.0.0/0",
    "createDate": "Dec 6, 2015 3:04:34 PM",
    "guestIp": "10.0.0.244",
    "lastOpDate": "Dec 6, 2015 3:04:34 PM",
    "name": "pf-9uf4",
    "privatePortEnd": 33,
    "privatePortStart": 33,
    "protocolType": "TCP",
    "state": "Enabled",
    "uuid": "310a6cd618144ca683d78d74307f16a4",
    "vipIp": "192.168.0.187",
    "vipPortEnd": 33,
    "vipPortStart": 33,
```

```
    "vipUuid": "433769b59a7c42199d762af01e08ec16",
    "vmNicUuid": "4b9c27321b794679a9ba8c18239bbb0d"
}
```

2.22.3 Operations

Create Port Forwarding Rule

Users can use CreatePortForwardingRule to create a port forwarding rule, with or without attaching to a VM nic. For example:

```
CreatePortForwardingRule name=pf1 vipPortStart=22 vipUuid=433769b59a7c42199d762af01e08ec16 protocolTy
```

A unattached rule can be attached to a VM nic later.

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
vipUuid	VIP UUID			0.6
vipPortStart	the start port of VIP		1 - 65535	0.6
vipPortEnd	the end port of VIP; if omitted, it's set to vipPortStart.	true	1 - 65535	0.6
privatePortStart	the start port of guest IP (VM nic's IP); if omitted, it's set to vipPortStart	true	1 - 65535	0.6
privatePortEnd	the end port for guest IP (VM nic's IP); if omitted, it's set to vipPortEnd	true	1 - 65535	0.6
protocolType	network traffic protocol type		• TCP • UDP	0.6
vmNicUuid	uuid of VM nic this port forwarding rule will be attached to	true		0.6
allowedCidr	source CIDR; the port forwarding rule only applies to traffics having this source CIDR; if omitted, it's set to 0.0.0.0/0	true		0.6

Delete Port Forwarding Rule

Users can use DeletePortForwardingRule to delete a port forwarding rule. For example:

```
DeletePortForwardingRule uuid=310a6cd618144ca683d78d74307f16a4
```

The VIP is recycled for other network services to use, if no more port forwarding rules bound to it.

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	rule uuid			0.6

Attach Port Forwarding Rule

Users can use AttachPortForwardingRule to attach a rule to a VM nic. For example:

```
AttachPortForwardingRule ruleUuid=310a6cd618144ca683d78d74307f16a4 vmNicUuid=4b9c27321b794679a9ba8c18
```

Parameters

Name	Description	Optional	Choices	Since
ruleUuid	rule uuid			0.6
vmNicUuid	VM nic uuid			0.6

Detach Port Forwarding Rule

Users can use DetachPortForwardingRule to detach a rule from a VM nic. For example:

```
DetachPortForwardingRule uuid=310a6cd618144ca683d78d74307f16a4
```

Parameters

Name	Description	Optional	Choices	Since
uuid	rule uuid			0.6

Query Port Forwarding Rule

Users can use QueryPortForwardingRule to query rules. For example:

```
QueryPortForwardingRule vipPortStart=22 vipIp=17.200.20.6
```

```
QueryPortForwardingRule vmNic.l3Network.name=database-tier
```

Primitive Fields

see *port forwarding rule inventory*

Nested And Expanded Fields

Field	Inventory	Description	Since
vip	*VIP inventory*	VIP this rule is bound	0.6
vmNic	*VM nic inventory*	VM nic this rule is attached	0.6

2.22.4 Global Configurations

snatInboundTraffic

Name	Category	Default Value	Choices
snatInboundTraffic	portForwarding	false	• true • false

Whether to source NAT inbound traffic of a port forwarding rule. If true, the traffics reaching portForwardingRule.guestIp will have a source IP equal to portForwardingRule.vipIp; this is useful when a VM has multiple port forwarding rules attached; it forces a VM to reply incoming traffics through VIPs where traffics come from, rather than replying through the default route.

2.22.5 Tags

Users can create user tags on a port forwarding rule with resourceType=PortForwardingRuleVO. For example:

```
CreateUserTag resourceType=PortForwardingRuleVO tag=ssh-rule resourceType=e960a93b7f974690bb779808f3c
```

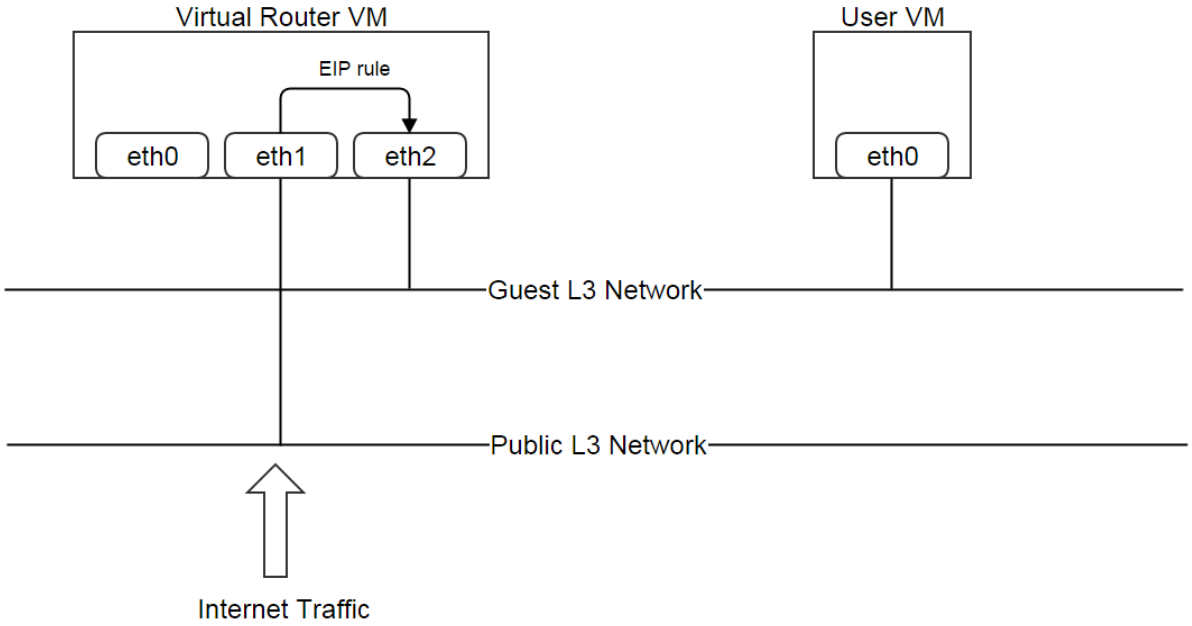
2.23 Elastic IP Address

Table of contents

- *Elastic IP Address*
  - *Overview*
  - *Inventory*
    - *Properties*
    - *Example*
  - *Operations*
    - *Create EIP*
      - *Parameters*
    - *Delete EIP*
      - *Parameters*
    - *Attach EIP*
      - *Parameters*
    - *Detach EIP*
      - *Parameters*
    - *Query EIP*
      - *Primitive Fields*
      - *Nested And Expanded Fields*
  - *Global Configurations*
    - *snatInboundTraffic*
  - *Tags*

2.23.1 Overview

An elastic IP(EIP) provides a way that allows outside network to reach a L3 network behind a source nat. EIP is based on network address translation(NAT) that maps an IP address of one network(usually a public network) to an IP address of another network(usually a private network); as being called elastic IP address, an EIP can be attached/detached to/from VMs dynamically.



2.23.2 Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
vmNicUuid	uuid of VM nic the EIP is bound	true		0.6
vipUuid	VIP uuid			0.6
state	EIP state, not implemented in this version		• Enabled • Disabled	0.6
vipIp	VIP IP address			0.6
guestIp	IP of VM nic	true		0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example

```
{
    "createDate": "Nov 28, 2015 6:52:14 PM",
    "guestIp": "10.0.0.170",
    "lastOpDate": "Nov 28, 2015 6:52:14 PM",
    "name": "eip-vlan10",
    "state": "Enabled",
    "uuid": "76b9231c94cd4a3aac497200bb26a643",
    "vipIp": "192.168.0.189",
    "vipUuid": "429106d5a63a4995911c2c5f14299b85",
    "vmNicUuid": "70cac1fd0c2f4940ba32645e09d3e22f"
}
```

2.23.3 Operations

Create EIP

Users can use CreateEip to create an EIP. For example:

```
CreateEip name=eip1 vipUuid=429106d5a63a4995911c2c5f14299b85 vmNicUuid=70cac1fd0c2f4940ba32645e09d3e2
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
vipUuid	VIP uuid			0.6
vmNicUuid	VM nic uuid; if omitted, the EIP is created without attaching to any VM nic.	true		0.6

Delete EIP

Users can use DeleteEip to delete an EIP. For example:

```
DeleteEip uuid=76b9231c94cd4a3aac497200bb26a643
```

After deleting, the VIP to which this EIP bound is recycled so other network services can reuse it.

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	EIP uuid			0.6

Attach EIP

Users can use AttachEip to attach an EIP to a VM nic. For example:

```
AttachEip eipUuid=76b9231c94cd4a3aac497200bb26a643 vmNicUuid=70cac1fd0c2f4940ba32645e09d3e22f
```

Parameters

Name	Description	Optional	Choices	Since
eipUuid	EIP uuid			0.6
vmNicUuid	VM nic uuid			0.6

Detach EIP

Users can use DetachEip to detach an EIP from the VM nic. For example:

```
DetachEip uuid=76b9231c94cd4a3aac497200bb26a643
```

Parameters

Name	Description	Optional	Choices	Since
uuid	EIP uuid			0.6

Query EIP

Users can use QueryEip to query EIPs. For example:

```
QueryEip vipIp=191.13.10.2
```

```
QueryEip vmNic.vmInstance.state=Running
```

Primitive Fields

see *EIP inventory*

Nested And Expanded Fields

Field	Inventory	Description	Since
vip	*VIP inventory*	VIP this EIP is bound	0.6
vmNic	*VM nic inventory*	VM nic is EIP is attached	0.6

2.23.4 Global Configurations

snatInboundTraffic

Name	Category	Default Value	Choices
snatInboundTraffic	eip	false	• true • false

Whether to source NAT inbound traffics of an EIP. If true, the traffics reaching eip.guestIp will have a source IP equal to eip.vipIp; this is useful when a VM has multiple EIP attached; it forces a VM to reply incoming traffic through the EIP where the traffic comes from, rather than replying through the default route.

2.23.5 Tags

Users can create user tags on an EIP with resourceType=EipVO. For example:

```
CreateUserTag resourceType=EipVO tag=web-public-ip resourceUuid=29fa6c2830c441aaa388d8165b80c24c
```

2.24 Volume Snapshot

Table of contents

- *Volume Snapshot*
  - *Overview*
    - *Snapshot Type*
    - *Snapshot Tree*
    - *Delete Snapshot*
  - *Volume Snapshot Tree Inventory*
    - *Properties*
      - *Example*
      - *Current*
      - *SnapshotLeafInventory*
      - *Properties*
  - *Volume Snapshot Inventory*
    - *Properties*
      - *Example*
      - *State*
      - *Status*
      - *VolumeSnapshotBackupStorageRefInventory*
      - *Properties*
  - *Operations*
    - *Create Snapshot*
      - *Parameters*
    - *Delete Snapshot*
      - *Parameters*
    - *Revert Volume From Snapshot*
      - *Parameters*
    - *Backup Snapshot*
      - *Parameters*
    - *Delete Snapshot Backup*
      - *Parameters*
    - *Create RootVolumeTemplate From Snapshot*
    - *Create Data Volume From Snapshot*
    - *Query Volume Snapshot*
      - *Primitive Fields*
      - *Nested And Expanded Fields*
  - *Global Configurations*
    - *incrementalSnapshot.maxNum*
    - *delete.parallelismDegree*
    - *backup.parallelismDegree*
  - *Tags*

2.24.1 Overview

A volume snapshot is a point-in-time capture of a VM's volume; memory and CPU state are not captured. Snapshots can be taken on root volumes and data volumes, and are arranged in a chain manner that the initial snapshot is usually a full snapshot containing all contents of a volume, and subsequent snapshots are delta snapshots only containing changes since the last snapshot. A volume can be restored to its old contents by reverting to a snapshot; images and volumes can be created from snapshots.

As volume snapshots only capture volumes' states, users need to flush changes in memory to file system in VMs' operating system before taking snapshots.

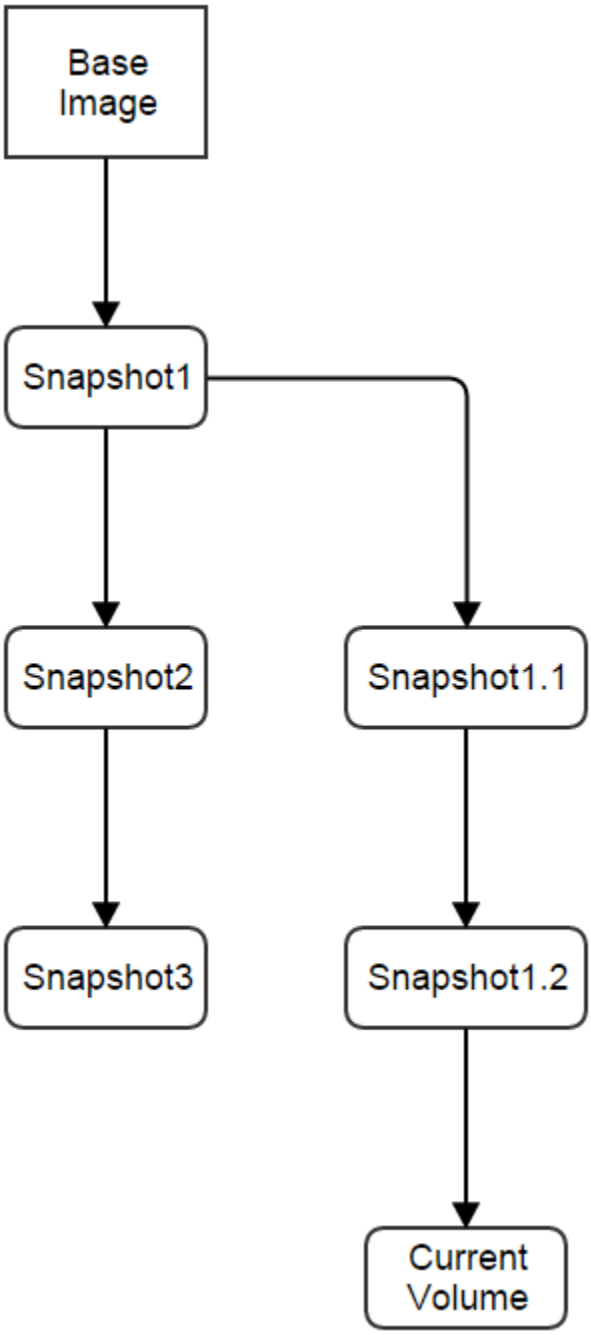
Snapshot Type

There are two ways to create volume snapshots; one is hypervisor based that snapshots are created by hypervisors from VMs' volumes; another is storage based that snapshots are created by storage systems that store VMs' volumes. In this ZStack version, only hypervisor based snapshot is supported.

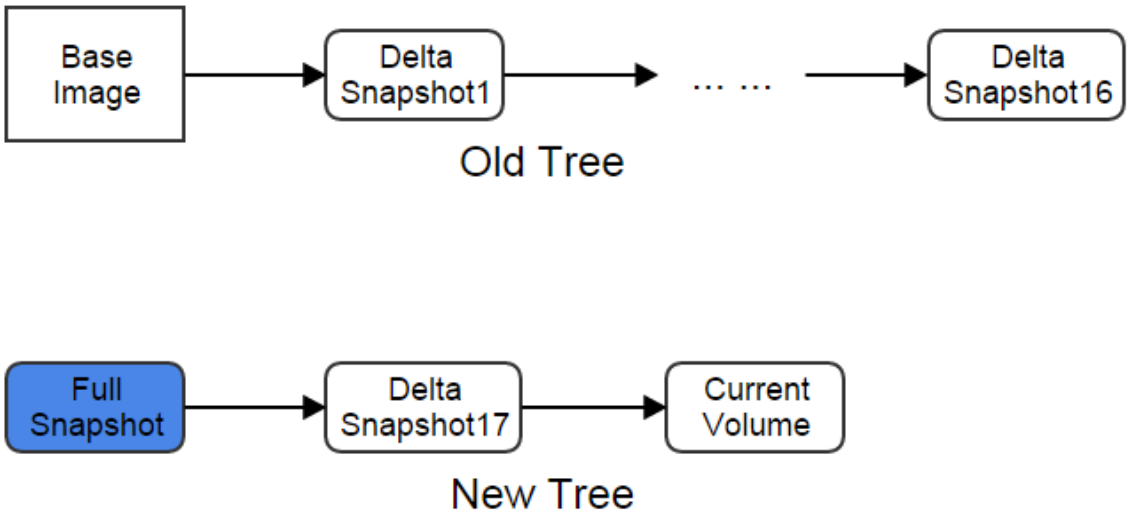
Snapshot Tree

Volume snapshots are normally arranged as a chain like:

however, once a volume is reverted to a snapshot and takes a snapshot again, the snapshot chain will grow as a tree where every chain is a branch:



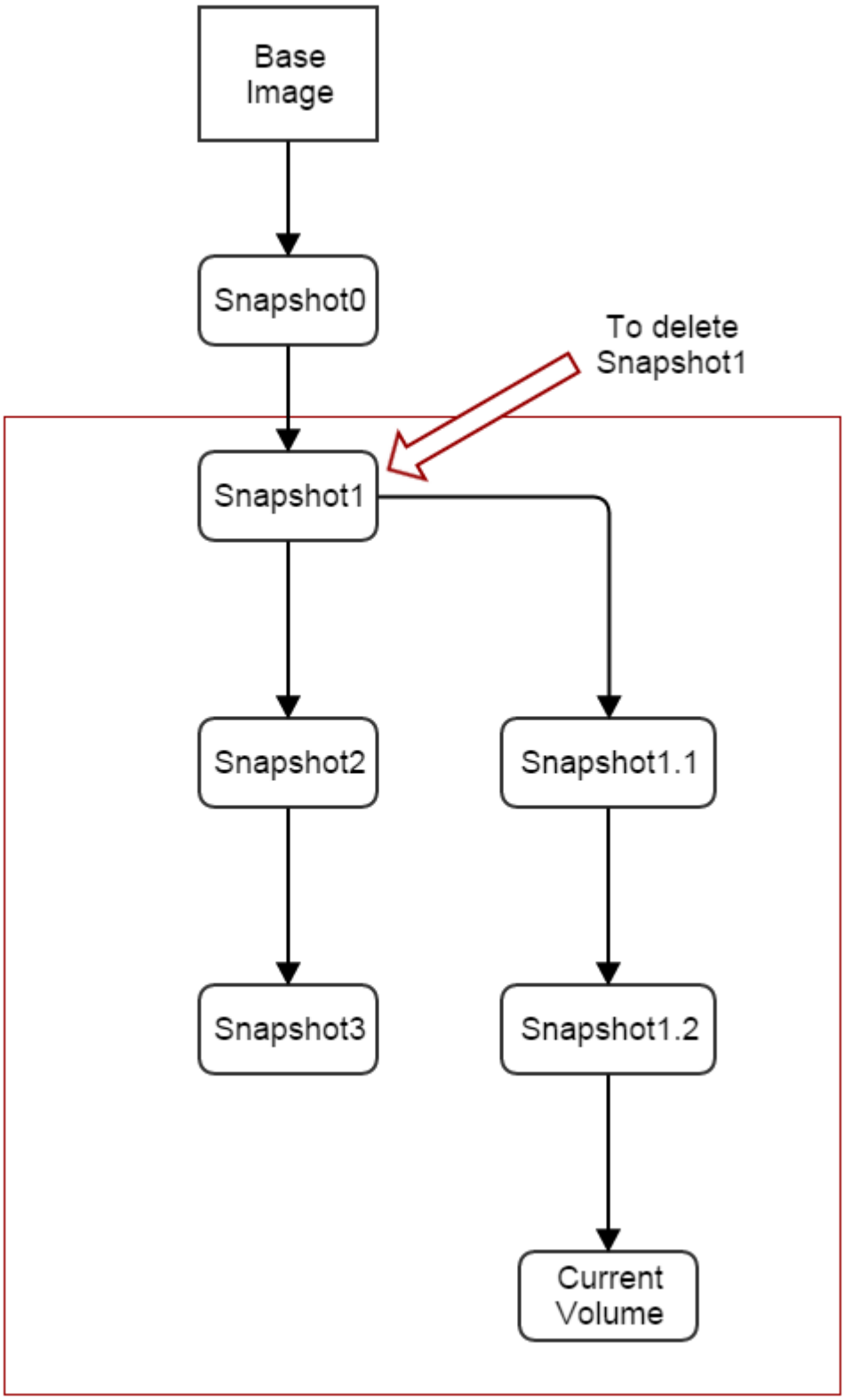
Current volume is always following the last snapshot; when a snapshot chain has too many delta snapshots, it may hurt the volume's disk IO performance, so ZStack sets max length of a snapshot chain to 16 by default; a new snapshot chain will be created after taking 16 snapshots.



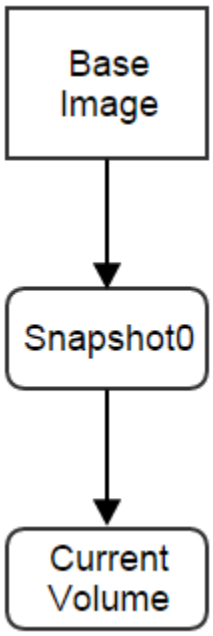
The max length of a snapshot chain can be configured by *incrementalSnapshot.maxNum*.

Delete Snapshot

When deleting a snapshot, if it's not a leaf that is the last one in a snapshot chain, all it's descendants will be deleted as well. For example:



after deleting Snapshot1, the Snapshot2, Snapshot3, Snapshot1.1, and Snapshot1.2 will deleted too, and the snapshot chain turns to be:



Note: When deleting a volume, all snapshots taken from this volume will be deleted from primary storage.

2.24.2 Volume Snapshot Tree Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
volumeUuid	the uuid of volume the snapshot tree is created			0.6
current	see *current*		• true • false	0.6
tree	a tree of *SnapshotLeafInventory*			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example

```
{
    "createDate": "Dec 7, 2015 11:45:02 PM",
    "current": true,
```

```
    "lastOpDate": "Dec 7, 2015 11:45:02 PM",
    "tree": {
        "children": [
            {
                "children": [
                    {
                        "children": [],
                        "inventory": {
                            "backupStorageRefs": [],
                            "createDate": "Dec 7, 2015 11:45:16 PM",
                            "format": "qcow2",
                            "lastOpDate": "Dec 7, 2015 11:45:16 PM",
                            "latest": true,
                            "name": "sp3",
                            "parentUuid": "3a859e89a39645018772e4d92ca02a09",
                            "primaryStorageInstallPath": "/opt/zstack/nfsprimarystorage/prim-a82b75ee
                            "primaryStorageUuid": "a82b75ee064a48708960f42b800bd910",
                            "size": 197120,
                            "state": "Enabled",
                            "status": "Ready",
                            "treeUuid": "acca6784c70b47fda68de18e2f8380d1",
                            "type": "Hypervisor",
                            "uuid": "b4d673e29f724320bb283c6dc4a59225",
                            "volumeType": "Root",
                            "volumeUuid": "2ad40ef516c540eeb138b7da24105f2e"
                        },
                        "parentUuid": "3a859e89a39645018772e4d92ca02a09"
                    }
                ],
                "inventory": {
                    "backupStorageRefs": [],
                    "createDate": "Dec 7, 2015 11:45:10 PM",
                    "format": "qcow2",
                    "lastOpDate": "Dec 7, 2015 11:45:10 PM",
                    "latest": false,
                    "name": "sp2",
                    "parentUuid": "b885d1e6549c49caab97322243827ca1",
                    "primaryStorageInstallPath": "/opt/zstack/nfsprimarystorage/prim-a82b75ee064a487
                    "primaryStorageUuid": "a82b75ee064a48708960f42b800bd910",
                    "size": 197120,
                    "state": "Enabled",
                    "status": "Ready",
                    "treeUuid": "acca6784c70b47fda68de18e2f8380d1",
                    "type": "Hypervisor",
                    "uuid": "3a859e89a39645018772e4d92ca02a09",
                    "volumeType": "Root",
                    "volumeUuid": "2ad40ef516c540eeb138b7da24105f2e"
                },
                "parentUuid": "b885d1e6549c49caab97322243827ca1"
            }
        ],
        "inventory": {
            "backupStorageRefs": [],
            "createDate": "Dec 7, 2015 11:45:02 PM",
            "format": "qcow2",
            "lastOpDate": "Dec 7, 2015 11:45:02 PM",
            "latest": false,
            "name": "sp1",
```

```
            "primaryStorageInstallPath": "/opt/zstack/nfsprimarystorage/prim-a82b75ee064a48708960f42b
            "primaryStorageUuid": "a82b75ee064a48708960f42b800bd910",
            "size": 4718592,
            "state": "Enabled",
            "status": "Ready",
            "treeUuid": "acca6784c70b47fda68de18e2f8380d1",
            "type": "Hypervisor",
            "uuid": "b885d1e6549c49caab97322243827ca1",
            "volumeType": "Root",
            "volumeUuid": "2ad40ef516c540eeb138b7da24105f2e"
        }
    },
    "uuid": "acca6784c70b47fda68de18e2f8380d1",
    "volumeUuid": "2ad40ef516c540eeb138b7da24105f2e"
}
```


Current

A current tree is a snapshot tree to which the volume currently links.

SnapshotLeafInventory

SnapshotLeafInventory is the leaf structure of snapshot tree; a snapshot tree always starts with a root SnapshotLeafInventory.

Properties

Name	Description	Optional	Choices	Since
inventory	the volume snapshot inventory, see *volume snapshot inventory*			0.6
parentUuid	uuid of *volume snapshot inventory* of parent leaf; if null, this leaf is the root leaf	true		0.6
children	a list of *SnapshotLeafInventory* which are child leafs			0.6

2.24.3 Volume Snapshot Inventory

Properties

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.6
name	see *Resource Properties*			0.6
description	see *Resource Properties*	true		0.6
type	see *type*		• Hypervisor • Storage	0.6
volumeUuid	uuid of volume the snapshot is created			0.6
treeUuid	the uuid of *tree* this snapshot belongs			0.6
parentUuid	uuid of parent snapshot in chain			0.6
primaryStorageUuid	uuid of primary storage this snapshot locates	true		0.6
primaryStorageInstallPath	the path of this snapshot on primary storage	true		0.6
volumeType	the type of volume this snapshot is created		• Root • Data	0.6
size	the snapshot size in bytes			0.6
state	snapshot state, see *state*		• Enabled • Disabled	0.6
status	snapshot status, see *status*		• Creating • Ready • Deleting	0.6
backupStorageRefs	a list of *VolumeSnapshotBackupStorageRefInventory*			0.6
createDate	see *Resource Properties*			0.6
lastOpDate	see *Resource Properties*			0.6

Example


```
{
    "backupStorageRefs": [],
    "createDate": "Dec 7, 2015 11:45:02 PM",
    "format": "qcow2",
    "lastOpDate": "Dec 7, 2015 11:45:02 PM",
    "latest": false,
    "name": "sp1",
    "primaryStorageInstallPath": "/opt/zstack/nfsprimarystorage/prim-a82b75ee064a48708960f42b800bd910
    "primaryStorageUuid": "a82b75ee064a48708960f42b800bd910",
    "size": 4718592,
    "state": "Enabled",
    "status": "Ready",
    "treeUuid": "acca6784c70b47fda68de18e2f8380d1",
    "type": "Hypervisor",
    "uuid": "b885d1e6549c49caab97322243827ca1",
    "volumeType": "Root",
    "volumeUuid": "2ad40ef516c540eeb138b7da24105f2e"
}
```


State

Volume snapshots have two states:

- **Enabled**

  The state allows operations to be proceeded

- **Disabled**

  The state that forbids operations; snapshots in this state cannot be used to revert volumes and create templates/volumes; and cannot be backup.

Status

Volume snapshots have following status:

- **Creating**

  The snapshot is being created from a volume

- **Ready**

  The snapshot is ready for any operations

- **Deleting**

  The snapshot is being deleted

VolumeSnapshotBackupStorageRefInventory

VolumeSnapshotBackupStorageRefInventory encompasses information about a copy of a snapshot on a backup storage.

Properties

Name	Description	Optional	Choices	Since
volumeSnapshotUuid	snapshot uuid			0.6
backupStorageUuid	backup storage uuid			0.6
installPath	the install path of snapshot copy on backup storage			0.6

2.24.4 Operations

Create Snapshot

Users can use CreateVolumeSnapshot to create a volume snapshot. For example:

```
CreateVolumeSnapshot name=sp1 volumeUuid=2ad40ef516c540eeb138b7da24105f2e
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.6
resourceUuid	resource uuid, see *Create Resources*	true		0.6
description	resource description, see *Resource Properties*	true		0.6
volumeUuid	volume uuid the snapshot is going to create			0.6

Delete Snapshot

Users can use DeleteVolumeSnapshot to delete a snapshot. For example:

```
DeleteVolumeSnapshot uuid=b885d1e6549c49caab97322243827ca1
```

Warning: All descendant snapshots will deleted as well. see *delete snapshot*

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	snapshot uuid			0.6

Revert Volume From Snapshot

Users can use RevertVolumeFromSnapshot to revert a volume to a snapshot; after reverting, the volume will have contents when the snapshot was created. For example:

```
RevertVolumeFromSnapshot uuid=b885d1e6549c49caab97322243827ca1
```

the volume is the one where the snapshot is created.

Parameters

Name	Description	Optional	Choices	Since
uuid	snapshot uuid			0.6

Backup Snapshot

Users can use BackupVolumeSnapshot to backup a snapshot to a backup storage. For example:

```
BackupVolumeSnapshot uuid=b885d1e6549c49caab97322243827ca1 backupStorageUuid=a82b75ee064a48708960f42
```

ancestor snapshots not backup on any backup storage will be backup as well.

Parameters

Name	Description	Optional	Choices	Since
uuid	snapshot uuid			0.6
backupStorageUuid	backup storage uuid; if omitted, ZStack will find a proper one.	true		0.6

Delete Snapshot Backup

Users can use DeleteVolumeSnapshotFromBackupStorage to delete a copy of snapshot from backup storage. For example:

```
DeleteVolumeSnapshotFromBackupStorage uuid=b885d1e6549c49caab97322243827ca1 backupStorageUuid=a82b75e
```

if the copy is the only copy of this snapshot on backup storage, all copies of descendant snapshots of this snapshot will be deleted as well;

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.6
uuid	snapshot uuid			0.6
backupStorageUuids	a list of uuid of backup storage from which to delete the snapshot's copy			0.6

Create RootVolumeTemplate From Snapshot

see *Create RootVolumeTemplate From Volume Snapshot*.

Create Data Volume From Snapshot

see *create data volume from volume snapshot*.

Query Volume Snapshot

Users can use QueryVolumeSnapshot to query volume snapshots. For example:

```
QueryVolumeSnapshot primaryStorageUuid=6572ce44c3f6422d8063b0fb262cbc62
```

```
QueryVolumeSnapshot volume.vmInstance.uuid=bd1652b1e44144e6b9b5b286b82edb69
```

Primitive Fields

see *volume snapshot inventory*

Nested And Expanded Fields

Field	Inventory	Description	Since
volume	*volume inventory*	the volume the volume snapshot is created	0.6
tree	*volume snapshot tree inventory*	the parent volume snapshot tree	0.6
primaryStorage	*primary storage inventory*	primary storage the volume snapshot locates	0.6
backupStorageRef	*VolumeSnapshotBackupStorageRefInventory*	the backup storage reference	0.6
backupStorage	*backup storage inventory*	backup storage that the volume snapshot locates	0.6

2.24.5 Global Configurations

incrementalSnapshot.maxNum

Name	Category	Default Value	Choices
incrementalSnapshot.maxNum	volumeSnapshot	16	> 0

The max length of a snapshot chain.

delete.parallelismDegree

Name	Category	Default Value	Choices
delete.parallelismDegree	volumeSnapshot	1	> 0

The number of snapshots that can be deleted in parallel when deleting a snapshot or a snpashot tree.

backup.parallelismDegree

Name	Category	Default Value	Choices
backup.parallelismDegree	volumeSnapshot	5	> 0

The number of snapshots that can be backup in parallel when backup snapshots.

2.24.6 Tags

Users can create user tags on a volume snapshot with resourceType=VolumeSnapshotVO. For example:

```
CreateUserTag resourceType=VolumeSnapshotVO tag=firstSnapshot resourceUuid=fae9a6f43c8e4017b0e2a251d
```

and create user tags on a volume snapshot tree with resourceType=VolumeSnapshotTreeVO. For example:

```
CreateUserTag resourceType=VolumeSnapshotVO tag=devops-tree resourceUuid=d6c49e73927d40abbfcf13852dc
```

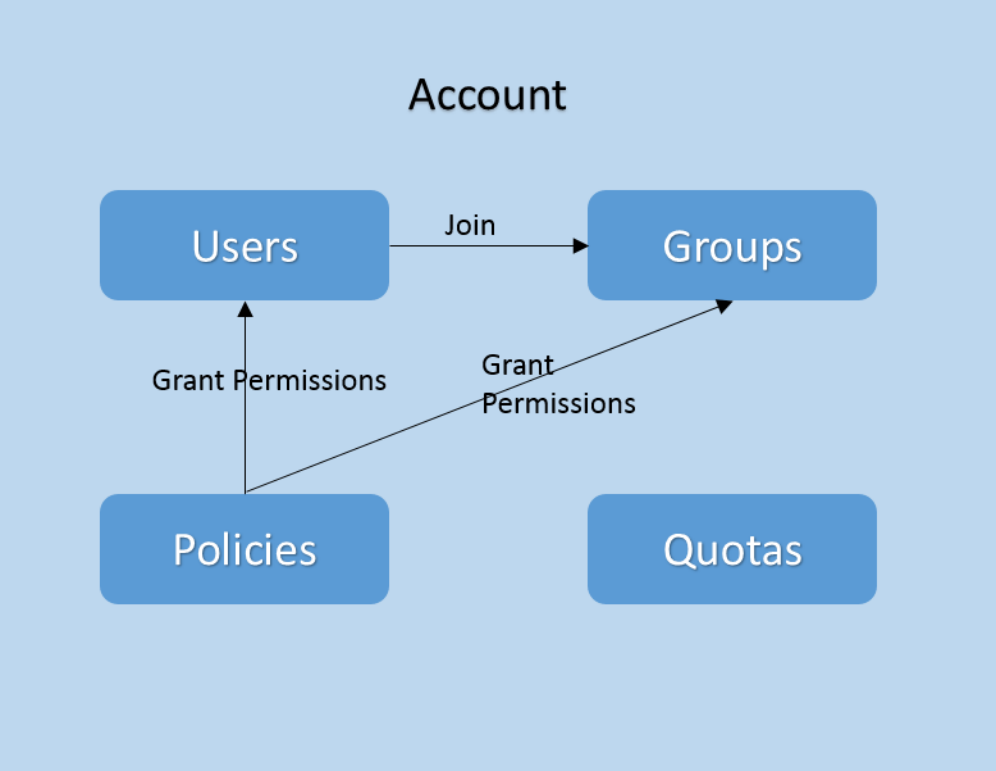
2.25 Identity

Table of contents

- *Identity*
  - *Overview*
    - *Account*
      - *Account Inventory*
      - *Example*
    - *Users*
      - *User Inventory*
      - *Example*
    - *Groups*
      - *Group Inventory*
      - *Example*
    - *Policies*
      - *Policy Inventory*
      - *Example*
      - *Statements:*
      - *Statement Inventory*
    - *Quota*
  - *Permission Control*
    - *Using users and groups*
    - *Permission Evaluation*
    - *Default Read Policy*
  - *Admin Account*
  - *Shared Resources*
  - *Operations*
    - *Create Account*
      - *Parameters*
    - *Create Users*
      - *Parameters*
    - *Create Groups*
      - *Parameters*
    - *Create Polices*
      - *Parameters*
    - *Add Users into Groups*
      - *Parameters*
    - *Attach Polices to Groups*
      - *Parameters*
    - *Attach Polices to Users*
      - *Parameters*
    - *Detach Polices from Groups*
      - *Parameters*
    - *Detach Polices from Users*
      - *Parameters*
    - *Reset Account Password*
      - *Parameters*
    - *Reset User Password*
      - *Parameters*
    - *Delete Groups*
      - *Parameters*
    - *Delete Users*
      - *Parameters*
    - *Delete Policies*
      - *Parameters*
    - *Delete Accounts*
      - *Parameters*
    - *Update Account Quota*
      - *Parameters*
    - *Share Resources*
      - *Parameters*

2.25.1 Overview

ZStack's identity service provides access control to ZStack resources for users. The system consists of concepts of account, user, group, policy, and quota. A global picture of the identity system is like:



Account

To manipulate resources, people need to create accounts that are the root identity to own all their resources. There are two types of accounts: admin and normal. Admin accounts, which have unlimited permissions, are owned by administrators. Normal accounts, which have only permissions to VM, instance offerings, disk offerings, L3 networks, images and so on, are created by admin accounts to allow people to manipulate those resources.

APIs are categorized in to admin-only APIs and non-admin APIs. A list of admin-only APIs can be found at *admin-only APIs*, and a list of non-admin APIs can be found at *non-admin APIs*.

Account Inventory

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.8
name	account name. see *Resource Properties*			0.8
description	see *Resource Properties*	true		0.8
createDate	see *Resource Properties*			0.8
lastOpDate	see *Resource Properties*			0.8

Note: The password will not be shown in the API returns for security reason.

Example

```
{
  "inventory": {
      "createDate": "Jul 22, 2015 10:18:34 AM",
      "lastOpDate": "Jul 22, 2015 10:18:34 AM",
      "name": "frank",
      "uuid": "3153a08ab21f46ca9e8b40ecfeec4255"
  }
}
```

Users

As an non-admin account has unlimited permissions to all resources it owns, people may create users to have finely-grained permission control. Users can only perform APIs assigned by policies.

User Inventory

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.8
name	user name, see *Resource Properties*			0.8
description	see *Resource Properties*	true		0.8
accountUuid	uuid of the owner account			0.8
createDate	see *Resource Properties*			0.8
lastOpDate	see *Resource Properties*			0.8

Note: The password will not be shown in the API returns for security reason.

Example

```
{
  "inventory": {
      "accountUuid": "36c27e8ff05c4780bf6d2fa65700f22e",
      "createDate": "Jul 22, 2015 10:21:50 AM",
      "lastOpDate": "Jul 22, 2015 10:21:50 AM",
      "name": "user1",
      "uuid": "68ebcf6260c94adab9dcce9e059e0025"
  }
}
```

Groups

Accounts can create groups to aggregate users. By assigning policies to groups, accounts can grant the same permissions to a group of users.

Group Inventory

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.8
name	group name, see *Resource Properties*			0.8
description	see *Resource Properties*	true		0.8
accountUuid	uuid of the owner account			0.8
createDate	see *Resource Properties*			0.8
lastOpDate	see *Resource Properties*			0.8

Example

```
{
  "inventory": {
      "accountUuid": "36c27e8ff05c4780bf6d2fa65700f22e",
      "createDate": "Jul 22, 2015 10:23:02 AM",
      "name": "group1",
      "uuid": "0939fc6f772d44d6a8f9d45c89c2a716"
  }
}
```

Policies

Polices are permissions that define what APIs users can perform. A policy consists of an array of *statements* each of which defines permissions to APIs.

Policy Inventory

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.8
name	policy name, see *Resource Properties*			0.8
description	see *Resource Properties*	true		0.8
accountUuid	uuid of the owner account, see *account*			0.8
statements	a list of *statements* defining API permissions			0.8
createDate	see *Resource Properties*			0.8
lastOpDate	see *Resource Properties*			0.8

Example

```
{
  "inventories": [
      {
          "accountUuid": "3153a08ab21f46ca9e8b40ecfeec4255",
          "name": "DEFAULT-READ-3153a08ab21f46ca9e8b40ecfeec4255",
          "statements": [
              {
                  "actions": [
                      ".*:read"
                  ],
                  "effect": "Allow",
```

```
                    "name": "read-permission-for-account-3153a08ab21f46ca9e8b40ecfeec4255"
                }
            ],
            "uuid": "b5169828533b47988a0d09f262b5769c"
        }
    ]
}
```

Statements:

A statement is a JSON text, containing a list of string matching API identities and an effect: *Allow* or *Deny*. A statement looks like:

```
{
    "actions": [
        ".*:read",
        "instance:APICreateVmInstanceMsg"
    ],
    "effect": "Allow",
    "name": "read-permission-for-account-3153a08ab21f46ca9e8b40ecfeec4255"
}
```

actions is a list of action strings that match one or more API identities. An API identity is a string in format of ***api_category:api_name*** that uniquely identifies an API. An action string can be a full identity like *instance:APICreateVmInstanceMsg* that only matches one API, or a regular expression that matches multiple APIs, for example, ***instance:.**** will match all APIs under the category ***instance***. Most of APIs have only one identity that is ***api_category:api_name***; some APIs have more identities so people can use regular expressions to match a group of APIs.

effect tells the decision when a action string matches an API call, allow or deny.

Note: In this version, all ***read*** APIs have an extra identity in format of ***api_category:read***, for example, instance:read. The read APIs are those not performing operations but getting information from ZStack. For example, all query APIs are read APIs; for example, the API APIQueryVmInstanceMsg has a default identity *instance:APIQueryVmInstanceMsg* and an extra identity *instance:read*.

A category may have many read APIs. For example, the VM category('instance') has APIQueryVmInstanceMsg, APIQueryVmNicMsg, APIGetVmAttachableDataVolumeMsg and so forth. They all have an API identity ***instance:read***. So a statement containing such action string can grant all read APIs in VM category to users and groups.

A list of API identities can be found at *API identities*.

Statement Inventory

Name	Description	Optional	Choices	Since
name	statement name			0.8
effect	permission decision		• Allow • Deny	0.8
actions	a list of strings to match API identities			0.8

Quota

Admin accounts can use quotas to limit how many resources non-admin accounts can create. When creating a non-admin account, ZStack automatically assigns default quotas to it, and admins can change them by the API *Update-Quota*. A list of default quotas can be found at *default quotas*.

2.25.2 Permission Control

The most exciting thing of identity system is that you can control API permissions, deciding what people can call what APIs. When people login into ZStack, depending on the way they login, they can get different permissions regarding APIs.

Administrators: When login as an admin account, the people can call any APIs.

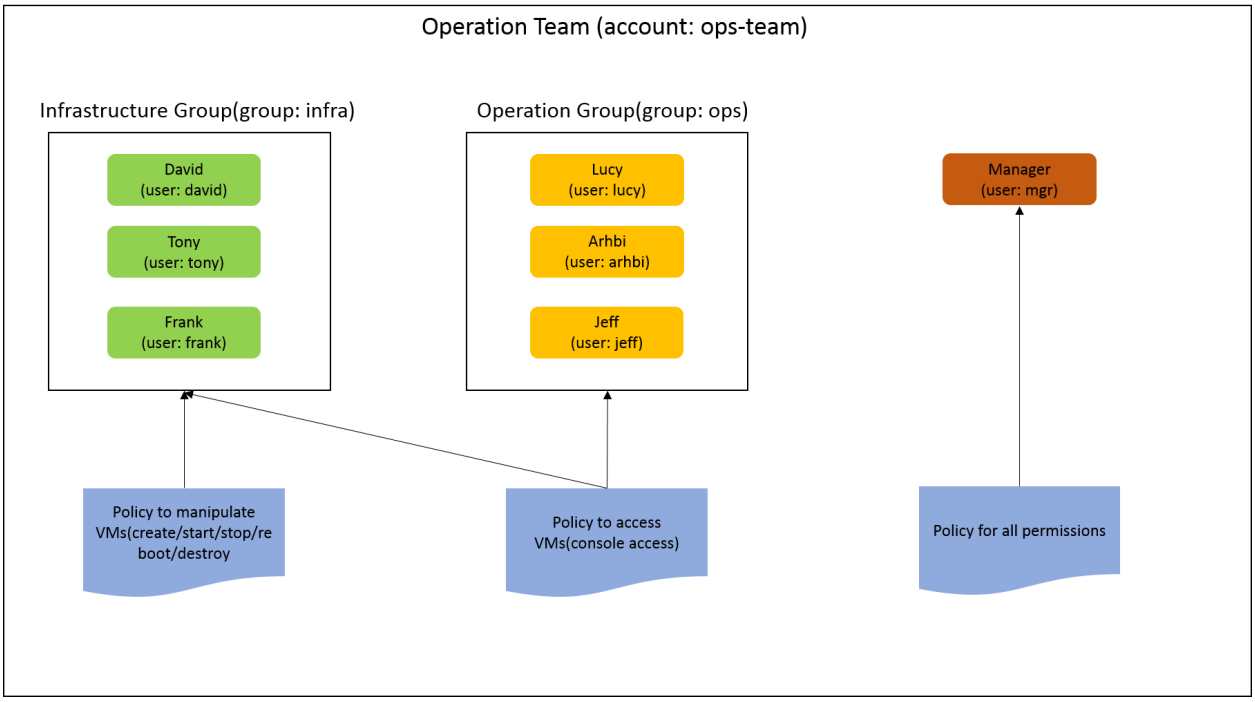
Non-admin Account: When login as a non-admin account, the people can perform any non-admin APIs.

User: When login as a user under an account, the people can only perform APIs granted by polices attached to the user or groups the user is in.

Using users and groups

The best way to limit people's permission in ZStack is only allowing they to login as users. Let's say you are a manager in a team that needs to apply some VMs in your company's IT infrastructure managed by ZStack. The first thing is to ask ZStack administrators in your company to create a non-admin account for you; once you get the account, you can create multiple users and groups with proper polices attached; then you can give those users to your team members , who can manipulate ZStack resources under the permissions you granted by polices.

An example helps to understand all those stuff, say you want to create below organization for your team:



In this organization, you have an infrastructure group responsible for managing VMs; the group has three members: David, Tony, Frank; you have another operation group for operating the VMs, which also has three users: Lucy, Arhbi, Jeff. The infrastructure group has permissions to manage VMs' lifecycle while the operation group can only use VMs

by accessing their consoles. In addition, you as the manager have all API permissions of your team's account(ops-team). To create such an organization:

Create the account ops-team:

```
>>>CreateAccount name=ops-team password=password
```

Note: make sure you login as the admin account to create the account

Login using the account ops-team:

```
>>>LogInByAccount accountName=ops-team password=password
```

Create users:

```
>>>CreateUser name=david password=password
```

repeat the step to create all users (tony, frank, lucy, arhbi, jeff, mgr)

Create groups:

```
>>>CreateUserGroup name=infra
```

repeat the step to create another group(ops)

Add users to groups:

```
>>>AddUserToGroup userUuid=d7646ae8af2140c0a3ccef2ad8da816d groupUuid=92c523a43651442489f8d2d598c7c3c
```

Note: The userUuid and groupUuid are printed on the screen when you create users and groups

repeat the step to add users into proper groups. infra group(david, tony, frank), ops group(lucy, arhbi, jeff).

Create polices

create the first policy allowing to call all VM related APIs:

```
>>>CreatePolicy name=vm-management statements='[{"actions":["instance:.*"], "effect":"Allow"}]'
```

create the second policy only allowing to access VM's console:

```
>>>CreatePolicy name=vm-console statements='[{"actions":["instance:APIRequestConsoleAccessMsg"], "ef
```

create the third policy allowing all APIs:

```
>>>CreatePolicy name=all statements='[{"actions":[".*"], "effect":"Allow"}]'
```

Warning: Please note the *statements* field is a JSON string encompassed by **single quotes**, and its contents are using **double quotes**. Please follow this convention otherwise the JSON string may not be able to be correctly parsed.

Attach policies to groups

attach the policy *vm-management* to the infrastructure group:

```
>>>AttachPolicyToUserGroup groupUuid=92c523a43651442489f8d2d598c7c3da policyUuid=afb3bfbb911a42e0a662
```

attach the policy *vm-console* to the operation group:

```
>>>AttachPolicyToUserGroup groupUuid=0939fc6f772d44d6a8f9d45c89c2a716 policyUuid=3bddf41e2ba6469881a
```

Note: The policyUuid and groupUuid are printed on the screen when you create groups and policies

Attach policies to user manager(mgr)

attach the policy *all* to the manager(user: mgr):

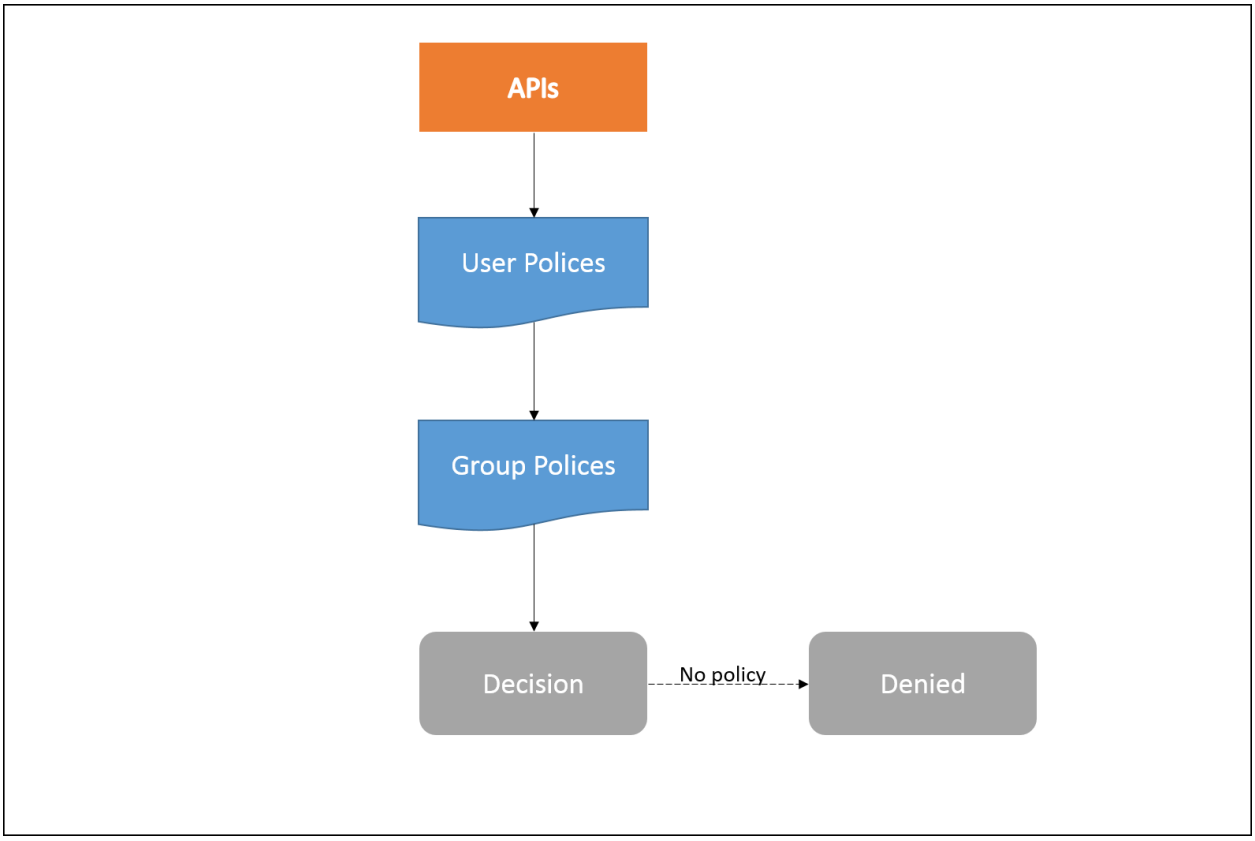
```
>>>AttachPolicyToUser userUuid=d55c5fba4d1b4533961db9952dc15b00 policyUuid=36c27e8ff05c4780bf6d2fa65
```

Note: The policyUuid and userUuid are printed on the screen when you create the policy and the user

Now your organization is created successfully, your team members can use user credentials to login.

Permission Evaluation

A policy consists of a list of statements each of which defines permissions(Allow or Deny) to APIs; users can have multiple polices attached either to themselves or to groups they are in. When users call APIs, it always evaluates from their polices then to group polices until a decision is made(Allow or Deny). If there is no policy matching an API, the API will be denied by default.



Default Read Policy

When creating a user, a default read policy **(action: .*:read, effect: Allow)** is attached to the new user so the user can query resources(e.g. VMs, L3 networks).

2.25.3 Admin Account

After installing ZStack, an admin account(account name: admin, password: password) is created by default. Administrators can use this account to create admin users which will have unlimited permissions just like the admin account, in order to allow different administrators to use own credentials to login. The password of the admin account can be changed by the API *UpdateAccount*.

2.25.4 Shared Resources

An account can share resources to other accounts. This is particularly useful in public clouds that the admin account can pre-defined some templates (e.g. images, instance offerings, disk offerings, l3 networks) so non-admin accounts(usually registered by customers) can use those templates to create VMs. See API *ShareResource*.

Resources can be shared to specified accounts or all accounts. When the API *ShareResource* is called with the parameter *toPublic* set to true, the resources specified in *resourceUuids* are shared to all accounts, otherwise they are shared to accounts specified in *accountUuids*. When you revoke the shared resources by the API *RevokeSharing*, you can specify *accountUuids* to revoke resources from certain accounts, or can set *toPublic* to true to revoke resources that have been shared to all accounts.

Note: In this version as the concept *role* has not been supported, other accounts can only read shared resources. That is to say, other accounts can query shared resources and use them (e.g. use images to create VMs) but cannot perform operations on them, for example, other accounts cannot delete a shared image.

2.25.5 Operations

Create Account

After login, the admin account can use CreateAccount create non-admin accounts. For example:

```
CreateAccount name=frank password=123456
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.8
resourceUuid	resource uuid, see *Create Resources*	true		0.8
description	resource description, see *Resource Properties*	true		0.8
name	account name			0.8
password	account password			0.8

Create Users

An account can user CreateUser to create a user. For example:

```
>>>CreateUser name=david password=123456
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.8
resourceUuid	resource uuid, see *Create Resources*	true		0.8
description	resource description, see *Resource Properties*	true		0.8
name	user name			0.8
password	user password			0.8

Create Groups

An account can use CreateUserGroup to create a group. For example:

```
>>>CreateUserGroup name=group
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.8
resourceUuid	resource uuid, see *Create Resources*	true		0.8
description	resource description, see *Resource Properties*	true		0.8
name	group name			0.8

Create Polices

An account can use CreatePolicy to create a policy. For example:

```
>>>CreatePolicy name=all statements='[{"actions":[".*"], "effect":"Allow"}]'
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.8
resourceUuid	resource uuid, see *Create Resources*	true		0.8
name	policy name			0.8
statements	a JSON string representing *statements*			0.8

Add Users into Groups

An account can use AddUserToGroup to add a user into a group. For example:

```
>>>AddUserToGroup userUuid=d7646ae8af2140c0a3ccef2ad8da816d groupUuid=92c523a43651442489f8d2d598c7c3c
```

Parameters

Name	Description	Optional	Choices	Since
userUuid	user uuid			0.8
groupUuid	group uuid			0.8

Attach Polices to Groups

An account can use AttachPolicyToUserGroup to attach a policy to a group. For example:

```
>>>AttachPolicyToUserGroup groupUuid=92c523a43651442489f8d2d598c7c3da policyUuid=afb3bfbb911a42e0a662
```

Parameters

Name	Description	Optional	Choices	Since
groupUuid	group uuid			0.8
policyUuid	policy uuid			0.8

Attach Polices to Users

An account can use AttachPolicyToUser to attach a policy to a user. For example:

```
>>>AttachPolicyToUser userUuid=d55c5fba4d1b4533961db9952dc15b00 policyUuid=36c27e8ff05c4780bf6d2fa657
```

Parameters

Name	Description	Optional	Choices	Since
userUuid	user uuid			0.8
policyUuid	policy uuid			0.8

Detach Polices from Groups

An account can use DetachPolicyFromUserGroup to detach a policy from a group. For example:

```
>>>DetachPolicyFromUserGroup groupUuid=f1a092c6914840c9895c564abbc55375 policyUuid=afb3bfbb911a42e0a6
```

Parameters

Name	Description	Optional	Choices	Since
groupUuid	group uuid			0.8
policyUuid	policy uuid			0.8

Detach Polices from Users

An account can use DetachPolicyFromUser to detach a policy from a user. For example:

```
>>>DetachPolicyFromUser policyUuid=36c27e8ff05c4780bf6d2fa65700f22e userUuid=d7646ae8af2140c0a3ccef2a
```

Parameters

Name	Description	Optional	Choices	Since
policyUuid	policy uuid			0.8
userUuid	user uuid			0.8

Reset Account Password

An account can use UpdateAccount to reset its password. For example:

```
>>>UpdateAccount password=password
```

Parameters

Name	Description	Optional	Choices	Since
password	the new password			0.8
uuid	the uuid of account to reset the password. It's mainly used by the admin account to reset passwords of other accounts. For non-admin accounts, this field is ignored as ZStack can figure out the account uuid by the current session.	true		0.8

Reset User Password

An account or a user can use UpdateUser to reset the password. For example:

```
>>>UpdateUser password=password
```

Parameters

Name	Description	Optional	Choices	Since
password	the new password			0.8
uuid	the user uuid. It's mainly used by the account to change passwords of users. For user changing own password, this field is ignored as ZStack can figure out the user uuid by the current session.	true		0.8

Delete Groups

An account can use DeleteUserGroup to delete a group. For example:

```
>>>DeleteUserGroup uuid=bb0e50fe0cfa4ec1af1835f9c210ae8e
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.8
uuid	group uuid			0.8

Delete Users

An account can use DeleteUser to delete a user. For example:

```
>>>DeleteUser uuid=fa4ec1af1835f9c210ae8e
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.8
uuid	user uuid			0.8

Delete Policies

An account can use DeletePolicy to delete a policy. For example:

```
>>>DeletePolicy uuid=bb0e50fe0cfa4ec1af1835f9c210ae8e
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.8
uuid	policy uuid			0.8

Delete Accounts

The admin account can use DeleteAccount to delete an non-admin account. For example:

```
>>>DeleteAccount uuid=bb0e50fe0cfa4ec1af1835f9c210ae8e
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.8
uuid	account uuid			0.8

Warning: After deleting, all resources owned by the account will be deleted as well

Update Account Quota

The admin account can use UpdateQuota to update an account's quotas. For example:

```
>>>UpdateQuota identityUuid=bb0e50fe0cfa4ec1af1835f9c210ae8e name=vm.num value=100
```

Parameters

Name	Description	Optional	Choices	Since
identityUuid	the account uuid			0.8
name	quota name		• vip.num • securityGroup.num • l3.num • portForwarding.num • vm.num • vm.cpuNum • vm.memorySize • volume.data.num • volume.capacity • eip.num	0.8

Share Resources

An account can use ShareResource to share resources to other accounts. For example:

```
ShareResource accountUuids=bb0e50fe0cfa4ec1af1835f9c210ae8e,bb0e50fe0cfa4ec1af1835f9c210ae8e resource
```

Parameters

Name	Description	Optional	Choices	Since
accountUuids	a list of account uuids to which the resources are shared. If omitted, the *toPublic* must be set to true	true		0.8
resourceUuids	a list of resource uuids			0.8
toPublic	if set to true, resources are shared to all accounts	true	• true • false	0.8

Revoke Shared Resources

An account can use RevokeResourceSharing to revoke shared resources from accounts. For example:

```
RevokeResourceSharing accountUuids=bb0e50fe0cfa4ec1af1835f9c210ae8e resourceUuids=b0662d80cc4945f8aba
```

```
RevokeResourceSharing all=true accountUuids=bb0e50fe0cfa4ec1af1835f9c210ae8e
```

```
RevokeResourceSharing resourceUuids=b0662d80cc4945f8abaf6d1096da9eb5 toPublic=true
```

Parameters

Name	Description	Optional	Choices	Since
accountUuids	the accounts from which the shared resources are revoked. When field *all* is set, this field is ignored, as the resources will be revoked from all accounts to which the resources have been shared.	true		0.6
resourceUuids	resources to be revoked from accounts			0.6
all	if set, the resources will be revoked from all accounts to which the resources have been shared.	true	• true • false	0.6
toPublic	if the resources are shared with 'toPublic = true' when calling ShareResource, this field must be also set to true when revoking.	true	• true • false	0.6

Query Accounts

An account can use QueryAccount query its own, or the admin account can query all accounts. For example:

```
>>>QueryAccount name=test
```

```
>>>QueryAccount group.name=group1
```

see account inventory

Field	Inventory	Description	Since
group	group inventory	child group	0.6
user	user inventory	child user	0.6
policy	policy inventory	child policy	0.6
quota		child quota	0.6

Query Users

An account can use QueryUser to query users. For example:

```
>>>QueryUser name=frank
```

```
>>>QueryUser name=frank policy.name=allow
```

see user inventory

Field	Inventory	Description	Since
account	see account inventory	the parent account	0.6
group	see group inventory	the group the user is in	0.6
policy	see policy inventory	the policy attached to the user	0.6

Query Policy

An account can use QueryPolicy to query policies. For example:

```
>>>QueryPolicy name=vm-management
```

```
>>>QueryPolicy user.name=frank
```

see policy inventory

Field	Inventory	Description	Since
account	see account inventory	the parent account	0.6
group	see group inventory	groups the policy attached	0.6
user	see user inventory	users the policy attached	0.6

Query Groups

An account can use QueryUserGroup to query groups. For example:

```
>>>QueryUserGroup name=group1
```

```
>>>QueryUserGroup user.name=frank
```

see group inventory

Field	Inventory	Description	Since
account	see account inventory	the parent account	0.6
user	see user inventory	users in the group	0.6
policy	see policy inventory	the policy attached to the group	0.6

2.25.6 Reference

Admin-only APIs

```
QueryGlobalConfig
GetGlobalConfig
UpdateGlobalConfig
GetHostAllocatorStrategies
GetCpuMemoryCapacity
ChangeInstanceOffering
IsReadyToGo
GetPrimaryStorageTypes
AttachPrimaryStorageToCluster
GetPrimaryStorageCapacity
UpdatePrimaryStorage
QueryPrimaryStorage
ChangePrimaryStorageState
SyncPrimaryStorageCapacity
DeletePrimaryStorage
ReconnectPrimaryStorage
DetachPrimaryStorageFromCluster
GetPrimaryStorageAllocatorStrategies
GetVolumeSnapshotTree
QueryBackupStorage
AttachBackupStorageToZone
GetBackupStorageTypes
ChangeBackupStorageState
GetBackupStorageCapacity
DetachBackupStorageFromZone
UpdateBackupStorage
DeleteBackupStorage
AddNetworkServiceProvider
AttachNetworkServiceProviderToL2Network
DetachNetworkServiceProviderFromL2Network
AttachL2NetworkToCluster
QueryL2VlanNetwork
CreateL2VlanNetwork
DetachL2NetworkFromCluster
DeleteL2Network
CreateL2NoVlanNetwork
UpdateL2Network
GetL2NetworkTypes
DeleteSearchIndex
SearchGenerateSqlTrigger
CreateSearchIndex
QueryManagementNode
CreateMessage
QueryCluster
DeleteCluster
UpdateCluster
```

```
CreateCluster
ChangeClusterState
CreateAccount
LogInByUser
SessionMessage
UpdateQuota
QueryAccount
LogInByAccount
ValidateSession
LogOut
UpdateZone
DeleteZone
CreateZone
QueryZone
ChangeZoneState
ChangeHostState
ReconnectHost
UpdateHost
DeleteHost
GetHypervisorTypes
QueryHost
QueryApplianceVm
AddIscsiFileSystemBackendPrimaryStorage
QueryIscsiFileSystemBackendPrimaryStorage
UpdateIscsiFileSystemBackendPrimaryStorage
AddLocalPrimaryStorage
UpdateKVMHost
AddKVMHost
AddNfsPrimaryStorage
QuerySftpBackupStorage
ReconnectSftpBackupStorage
UpdateSftpBackupStorage
AddSftpBackupStorage
```

Non-admin APIs

```
UpdateVmInstance
GetVmAttachableL3Network
MigrateVm
StopVmInstance
GetVmAttachableDataVolume
QueryVmNic
AttachL3NetworkToVm
DestroyVmInstance
GetVmMigrationCandidateHosts
QueryVmInstance
DetachL3NetworkFromVm
RebootVmInstance
CreateVmInstance
StartVmInstance
ChangeImageState
UpdateImage
DeleteImage
CreateDataVolumeTemplateFromVolume
CreateRootVolumeTemplateFromRootVolume
QueryImage
CreateRootVolumeTemplateFromVolumeSnapshot
```

```
AddImage
RequestConsoleAccess
BackupDataVolume
AttachDataVolumeToVm
UpdateVolume
QueryVolume
CreateDataVolumeFromVolumeSnapshot
CreateDataVolumeFromVolumeTemplate
DetachDataVolumeFromVm
CreateDataVolume
GetDataVolumeAttachableVm
GetVolumeFormat
DeleteDataVolume
CreateVolumeSnapshot
ChangeVolumeState
DeleteDiskOffering
QueryInstanceOffering
UpdateInstanceOffering
CreateInstanceOffering
CreateDiskOffering
DeleteInstanceOffering
ChangeInstanceOfferingState
QueryDiskOffering
UpdateDiskOffering
ChangeDiskOfferingState
QueryVolumeSnapshotTree
DeleteVolumeSnapshot
UpdateVolumeSnapshot
DeleteVolumeSnapshotFromBackupStorage
QueryVolumeSnapshot
RevertVolumeFromSnapshot
BackupVolumeSnapshot
AddDnsToL3Network
CreateL3Network
GetFreeIp
UpdateL3Network
DeleteIpRange
ChangeL3NetworkState
AddIpRange
GetL3NetworkTypes
AddIpRangeByNetworkCidr
QueryIpRange
RemoveDnsFromL3Network
GetIpAddressCapacity
DeleteL3Network
UpdateIpRange
QueryL3Network
AttachNetworkServiceToL3Network
QueryNetworkServiceL3NetworkRef
QueryNetworkServiceProvider
GetNetworkServiceTypes
QueryL2Network
QueryUserTag
QuerySystemTag
DeleteTag
CreateUserTag
CreateSystemTag
QueryTag
```

```
AttachPolicyToUserGroup
RemoveUserFromGroup
AttachPolicyToUser
UpdateUser
AddUserToGroup
QueryQuota
ShareResource
DeleteAccount
CreateUserGroup
CreateUser
DetachPolicyFromUserGroup
QueryPolicy
QueryUser
DeletePolicy
RevokeResourceSharing
UpdateAccount
DeleteUser
DeleteUserGroup
CreatePolicy
DetachPolicyFromUser
QueryUserGroup
ReconnectVirtualRouter
QueryVirtualRouterOffering
CreateVirtualRouterOffering
QueryVirtualRouterVm
AttachPortForwardingRule
DetachPortForwardingRule
GetPortForwardingAttachableVmNics
ChangePortForwardingRuleState
UpdatePortForwardingRule
CreatePortForwardingRule
QueryPortForwardingRule
DeletePortForwardingRule
DetachEip
GetEipAttachableVmNics
UpdateEip
QueryEip
ChangeEipState
DeleteEip
CreateEip
AttachEip
ChangeSecurityGroupState
DetachSecurityGroupFromL3Network
DeleteSecurityGroupRule
CreateSecurityGroup
QueryVmNicInSecurityGroup
QuerySecurityGroup
AddSecurityGroupRule
QuerySecurityGroupRule
DeleteSecurityGroup
UpdateSecurityGroup
DeleteVmNicFromSecurityGroup
GetCandidateVmNicForSecurityGroup
AttachSecurityGroupToL3Network
AddVmNicToSecurityGroup
DeleteVip
UpdateVip
ChangeVipState
```

```
CreateVip
QueryVip
```

API Identities

```
ReconnectVirtualRouter: virtualRouter:APIReconnectVirtualRouterMsg

GetNetworkServiceProvider: l2Network:read, l2Network:APIGetNetworkServiceProviderMsg

AddDnsToL3Network: l3Network:APIAddDnsToL3NetworkMsg

DeleteSecurityGroup: securityGroup:APIDeleteSecurityGroupMsg

AddImage: image:APIAddImageMsg

QueryUser: identity:read, identity:APIQueryUserMsg

GetL3NetworkTypes: l3Network:read, l3Network:APIGetL3NetworkTypesMsg

ShareResource: identity:APIShareResourceMsg

QueryVirtualRouterOffering: virtualRouter:read, virtualRouter:APIQueryVirtualRouterOfferingMsg

QueryIpRange: l3Network:read, l3Network:APIQueryIpRangeMsg

AttachDataVolumeToVm: volume:APIAttachDataVolumeToVmMsg

QueryUserGroup: identity:read, identity:APIQueryUserGroupMsg

QueryVmNicInSecurityGroup: securityGroup:read, securityGroup:APIQueryVmNicInSecurityGroupMsg

CreateSystemTag: tag:APICreateSystemTagMsg

CreateVip: vip:APICreateVipMsg

DeleteDiskOffering: configuration:APIDeleteDiskOfferingMsg

StartVmInstance: instance:APIStartVmInstanceMsg

GetVmAttachableL3Network: instance:read, instance:APIGetVmAttachableL3NetworkMsg

DeleteVip: vip:APIDeleteVipMsg

GetDataVolumeAttachableVm: volume:read, volume:APIGetDataVolumeAttachableVmMsg

QuerySystemTag: tag:read, tag:APIQuerySystemTagMsg

AttachL3NetworkToVm: instance:APIAttachL3NetworkToVmMsg

CreateUserTag: tag:APICreateUserTagMsg

CreateVmInstance: instance:APICreateVmInstanceMsg

CreateSecurityGroup: securityGroup:APICreateSecurityGroupMsg

UpdateVolumeSnapshot: volumeSnapshot:APIUpdateVolumeSnapshotMsg
```

```
QueryDiskOffering: configuration:read, configuration:APIQueryDiskOfferingMsg

StopVmInstance: instance:APIStopVmInstanceMsg

CreateEip: eip:APICreateEipMsg

ChangePortForwardingRuleState: portForwarding:APIChangePortForwardingRuleStateMsg

UpdateL3Network: l3Network:APIUpdateL3NetworkMsg

ChangeDiskOfferingState: configuration:APIChangeDiskOfferingStateMsg

MigrateVm: instance:APIMigrateVmMsg

ChangeVipState: vip:APIChangeVipStateMsg

AddIpRange: l3Network:APIAddIpRangeMsg

CreateDataVolume: volume:APICreateDataVolumeMsg

CreateDataVolumeFromVolumeSnapshot: volume:APICreateDataVolumeFromVolumeSnapshotMsg

UpdateImage: image:APIUpdateImageMsg

QueryVmNic: instance:read, instance:APIQueryVmNicMsg

QueryTag: tag:read, tag:APIQueryTagMsg

GetPortForwardingAttachableVmNics: portForwarding:APIGetPortForwardingAttachableVmNicsMsg

DeleteInstanceOffering: configuration:APIDeleteInstanceOfferingMsg

AttachPortForwardingRule: portForwarding:APIAttachPortForwardingRuleMsg

DeletePortForwardingRule: portForwarding:APIDeletePortForwardingRuleMsg

CreatePortForwardingRule: portForwarding:APICreatePortForwardingRuleMsg

UpdateIpRange: l3Network:APIUpdateIpRangeMsg

GetFreeIp: l3Network:read, l3Network:APIGetFreeIpMsg

ChangeL3NetworkState: l3Network:APIChangeL3NetworkStateMsg

QueryVip: vip:read, vip:APIQueryVipMsg

UpdateEip: eip:APIUpdateEipMsg

QueryVolumeSnapshotTree: volumeSnapshot:read, volumeSnapshot:APIQueryVolumeSnapshotTreeMsg

DetachDataVolumeFromVm: volume:APIDetachDataVolumeFromVmMsg

RebootVmInstance: instance:APIRebootVmInstanceMsg

UpdateInstanceOffering: configuration:APIUpdateInstanceOfferingMsg

DestroyVmInstance: instance:APIDestroyVmInstanceMsg
```

```
UpdateUser: identity:APIUpdateUserMsg

QueryNetworkServiceL3NetworkRef: l3Network:read, l3Network:APIQueryNetworkServiceL3NetworkRefMsg

CreateL3Network: l3Network:APICreateL3NetworkMsg

GetNetworkServiceTypes: l3Network:read, l3Network:APIGetNetworkServiceTypesMsg

GetVmAttachableDataVolume: instance:read, instance:APIGetVmAttachableDataVolumeMsg

QueryL3Network: l3Network:read, l3Network:APIQueryL3NetworkMsg

CreateDataVolumeTemplateFromVolume: image:APICreateDataVolumeTemplateFromVolumeMsg

DeleteSecurityGroupRule: securityGroup:APIDeleteSecurityGroupRuleMsg

QueryUserTag: tag:read, tag:APIQueryUserTagMsg

DeleteVolumeSnapshotFromBackupStorage: volumeSnapshot:APIDeleteVolumeSnapshotFromBackupStorageMsg

CreateDiskOffering: configuration:APICreateDiskOfferingMsg

QuerySecurityGroup: securityGroup:read, securityGroup:APIQuerySecurityGroupMsg

QueryVolumeSnapshot: volumeSnapshot:read, volumeSnapshot:APIQueryVolumeSnapshotMsg

QueryPortForwardingRule: portForwarding:read, portForwarding:APIQueryPortForwardingRuleMsg

UpdateDiskOffering: configuration:APIUpdateDiskOfferingMsg

GetCandidateVmNicForSecurityGroup: securityGroup:read, securityGroup:APIGetCandidateVmNicForSecurityG

QueryPolicy: identity:read, identity:APIQueryPolicyMsg

GetEipAttachableVmNics: eip:APIGetEipAttachableVmNicsMsg

CreateInstanceOffering: configuration:APICreateInstanceOfferingMsg

AddIpRangeByNetworkCidr: l3Network:APIAddIpRangeByNetworkCidrMsg

UpdateVmInstance: instance:APIUpdateVmInstanceMsg

QueryVirtualRouterVm: virtualRouter:read, virtualRouter:APIQueryVirtualRouterVmMsg

RequestConsoleAccess: console:APIRequestConsoleAccessMsg

ChangeEipState: eip:APIChangeEipStateMsg

QuerySecurityGroupRule: securityGroup:read, securityGroup:APIQuerySecurityGroupRuleMsg

DetachSecurityGroupFromL3Network: securityGroup:APIDetachSecurityGroupFromL3NetworkMsg

CreateDataVolumeFromVolumeTemplate: volume:APICreateDataVolumeFromVolumeTemplateMsg

DeleteDataVolume: volume:APIDeleteDataVolumeMsg

AddVmNicToSecurityGroup: securityGroup:APIAddVmNicToSecurityGroupMsg
```

```
DeleteVolumeSnapshot: volumeSnapshot:APIDeleteVolumeSnapshotMsg

DetachEip: eip:APIDetachEipMsg

DetachPortForwardingRule: portForwarding:APIDetachPortForwardingRuleMsg

CreateVirtualRouterOffering: virtualRouter:APICreateVirtualRouterOfferingMsg

RevertVolumeFromSnapshot: volumeSnapshot:APIRevertVolumeFromSnapshotMsg

DeleteIpRange: l3Network:APIDeleteIpRangeMsg

UpdateVip: vip:APIUpdateVipMsg

AttachNetworkServiceToL3Network: l3Network:APIAttachNetworkServiceToL3NetworkMsg

DeleteTag: tag:APIDeleteTagMsg

RemoveDnsFromL3Network: l3Network:APIRemoveDnsFromL3NetworkMsg

DeleteL3Network: l3Network:APIDeleteL3NetworkMsg

UpdatePortForwardingRule: portForwarding:APIUpdatePortForwardingRuleMsg

ChangeVolumeState: volume:APIChangeVolumeStateMsg

QueryVmInstance: instance:read, instance:APIQueryVmInstanceMsg

GetVmMigrationCandidateHosts: instance:read, instance:APIGetVmMigrationCandidateHostsMsg

UpdateVolume: volume:APIUpdateVolumeMsg

QueryL2Network: l2Network:read, l2Network:APIQueryL2NetworkMsg

BackupVolumeSnapshot: volumeSnapshot:APIBackupVolumeSnapshotMsg

QueryQuota: identity:read, identity:APIQueryQuotaMsg

QueryImage: image:read, image:APIQueryImageMsg

RevokeResourceSharing: identity:APIRevokeResourceSharingMsg

UpdateSecurityGroup: securityGroup:APIUpdateSecurityGroupMsg

ChangeImageState: image:APIChangeImageStateMsg

AddSecurityGroupRule: securityGroup:APIAddSecurityGroupRuleMsg

QueryVolume: volume:read, volume:APIQueryVolumeMsg

AttachSecurityGroupToL3Network: securityGroup:APIAttachSecurityGroupToL3NetworkMsg

DeleteEip: eip:APIDeleteEipMsg

QueryEip: eip:read, eip:APIQueryEipMsg

DeleteImage: image:APIDeleteImageMsg
```

```
GetIpAddressCapacity: l3Network:read, l3Network:APIGetIpAddressCapacityMsg

ChangeInstanceOfferingState: configuration:APIChangeInstanceOfferingStateMsg

DeleteVmNicFromSecurityGroup: securityGroup:APIDeleteVmNicFromSecurityGroupMsg

CreateVolumeSnapshot: volumeSnapshot:APICreateVolumeSnapshotMsg

CreateRootVolumeTemplateFromRootVolume: image:APICreateRootVolumeTemplateFromRootVolumeMsg

GetVolumeFormat: volume:read, volume:APIGetVolumeFormatMsg

BackupDataVolume: volume:APIBackupDataVolumeMsg

CreateRootVolumeTemplateFromVolumeSnapshot: image:APICreateRootVolumeTemplateFromVolumeSnapshotMsg

QueryInstanceOffering: configuration:read, configuration:APIQueryInstanceOfferingMsg

ChangeSecurityGroupState: securityGroup:APIChangeSecurityGroupStateMsg

QueryNetworkServiceProvider: l3Network:read, l3Network:APIQueryNetworkServiceProviderMsg

AttachEip: eip:APIAttachEipMsg

DetachL3NetworkFromVm: instance:APIDetachL3NetworkFromVmMsg
```

Default Quotas

Name	Description	Value	Since
vip.num	max number of VIPs	20	0.8
securityGroup.num	max number of security groups	20	0.8
l3.num	max number of L3 networks	20	0.8
portForwarding.num	max number of port forwarding rules	20	0.8
vm.num	max number of VMs	20	0.8
vm.cpuNum	max number of VCPU cores	80	0.8
vm.memorySize	total size of memory	85899345920 bytes (80G)	0.8
volume.data.num	max number of data volumes	40	0.8
volume.capacity	total volume capacity of both data volumes and root volumes	10995116277760 bytes (10T)	0.8
eip.num	max number of EIPs	20	0.8

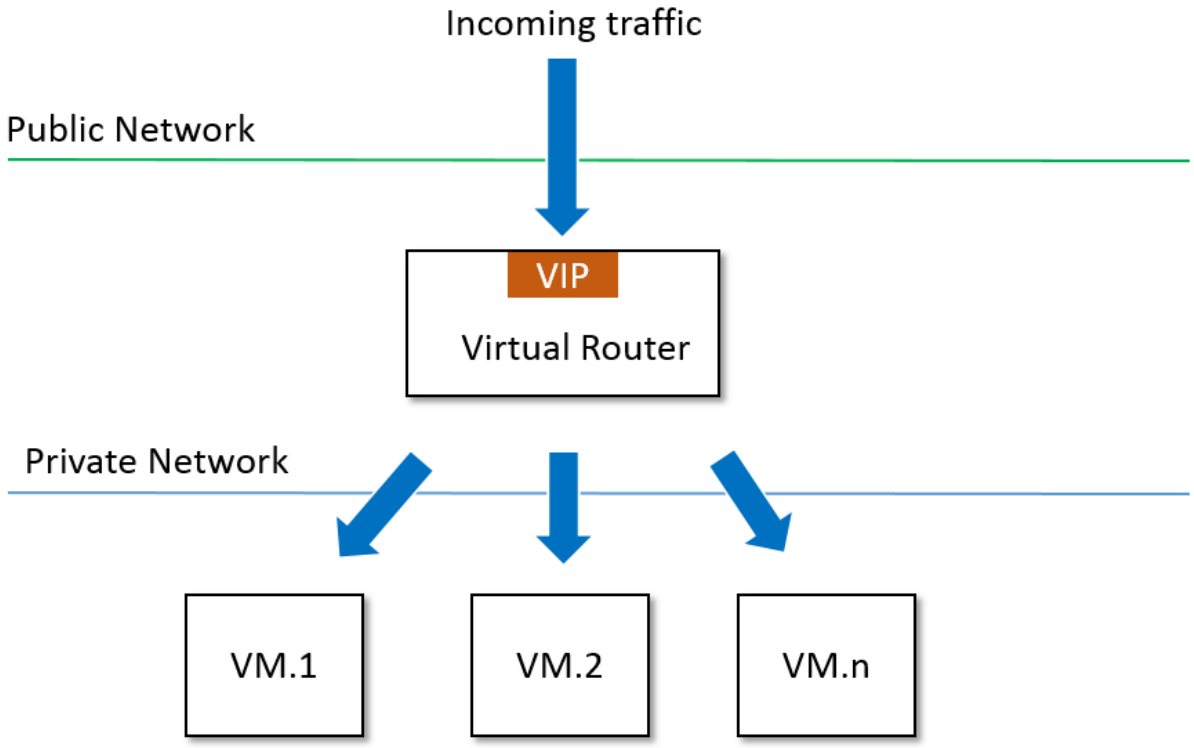
2.26 Elastic Load Balancer

Table of contents

- Elastic Load Balancer
  - Overview
  - Load Balancer
    - Inventory
    - Example
  - Listener
    - Inventory
    - Protocol
    - Example
  - Backend VM Nics
    - Nic Reference Inventory
  - A Full Example
  - Operations
    - Create Load Balancer
      - Parameters
    - Delete Load Balancer
      - Parameters
    - Create Listener
      - Parameters
    - Delete Listener
      - Parameters
    - Add VM Nic to Load Balancer
      - Parameters
    - Remove VM Nic from Load Balancer
      - Parameters
    - Query Load Balancer
      - Primitive Fields of Query
      - Nested And Expanded Fields of Query
    - Query Listener
      - Primitive Fields of Query
      - Nested And Expanded Fields of Query
  - Tags
    - System Tags
      - Separate Virtual Router
      - Listener Configurations
      - Healthy Threshold
      - Health Check Interval
      - Unhealthy Threshold
      - Connection Idle Timeout
      - Max Connections
      - Balancing Algorithm
  - Global Configurations
    - Connection Idle Timeout
    - Healthy Threshold
    - Unhealthy Threshold
    - Health Check Interval
    - Max Connection
    - Balancing Algorithm

2.26.1 Overview

Elastic Load Balancing automatically distributes your incoming application traffic across multiple VM instances. It detects unhealthy instances and reroutes traffic to healthy instances until the unhealthy instances have been restored.



2.26.2 Load Balancer

A load balancer consists of a *VIP* to which incoming traffics visit, a set of listeners that defines a variety of properties such as load balancer port, instance port, health-check configurations, and a group of VM nics where the incoming traffics will be routed.

Inventory

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.9
name	see *Resource Properties*			0.9
description	see *Resource Properties*	true		0.9
state	reserved in 0.9 version, always Enabled		• Enabled • Disabled	0.9
vipUuid	uuid of *VIP*			0.9
listeners	a list of *listener*			0.9
createDate	see *Resource Properties*			0.9
lastOpDate	see *Resource Properties*			0.9

Example


```
{
    "listeners": [
        {
            "createDate": "Aug 20, 2015 2:54:14 PM",
            "instancePort": 80,
            "lastOpDate": "Aug 20, 2015 2:54:14 PM",
            "loadBalancerPort": 80,
            "loadBalancerUuid": "0188cec6635845e0b2526a8e7e090e2a",
            "name": "80",
            "protocol": "http",
            "uuid": "ba5f192472ab4fc4b36e5af873f0fec5",
            "vmNicRefs": [
                {
                    "createDate": "Aug 20, 2015 2:55:49 PM",
                    "id": 18,
                    "lastOpDate": "Aug 20, 2015 2:55:49 PM",
                    "listenerUuid": "ba5f192472ab4fc4b36e5af873f0fec5",
                    "status": "Active",
                    "vmNicUuid": "35b8aadef2f847d9836bdf06121e1c29"
                },
                {
                    "createDate": "Aug 20, 2015 2:55:49 PM",
                    "id": 19,
                    "lastOpDate": "Aug 20, 2015 2:55:49 PM",
                    "listenerUuid": "ba5f192472ab4fc4b36e5af873f0fec5",
                    "status": "Active",
                    "vmNicUuid": "df7d40a47cb640a9b40001f2f318989a"
                }
            ]
        },
        {
            "createDate": "Aug 20, 2015 5:29:39 AM",
            "instancePort": 22,
            "lastOpDate": "Aug 20, 2015 5:29:39 AM",
```

```
            "loadBalancerPort": 22,
            "loadBalancerUuid": "0188cec6635845e0b2526a8e7e090e2a",
            "name": "ssh",
            "protocol": "tcp",
            "uuid": "2901fd13765c492b9a3d004e806a0beb",
            "vmNicRefs": [
                {
                    "createDate": "Aug 20, 2015 5:30:07 AM",
                    "id": 15,
                    "lastOpDate": "Aug 20, 2015 5:30:07 AM",
                    "listenerUuid": "2901fd13765c492b9a3d004e806a0beb",
                    "status": "Active",
                    "vmNicUuid": "35b8aadef2f847d9836bdf06121e1c29"
                },
                {
                    "createDate": "Aug 20, 2015 5:30:07 AM",
                    "id": 16,
                    "lastOpDate": "Aug 20, 2015 5:30:07 AM",
                    "listenerUuid": "2901fd13765c492b9a3d004e806a0beb",
                    "status": "Active",
                    "vmNicUuid": "df7d40a47cb640a9b40001f2f318989a"
                }
            ]
        }
    ],
    "name": "lb",
    "state": "Enabled",
    "uuid": "0188cec6635845e0b2526a8e7e090e2a",
    "vipUuid": "df6a73601f1741fd847cf5456b0d42ac"
}
```


2.26.3 Listener

A listener defines how the load balancer routes incoming traffics from a VIP port(called loadBalancer port) to a backend port(called instancePort) of VM instances, and a set of properties that how the load balancer should handle stuff like connection timeout, health-check threshold.

From users' perspective, they create a listener whenever they want to load balance traffics from a frontend port(loadBalancerPort) on the load balancer to a backend port(instancePort) of VM instances running on a private network.

A load balancer can have many listeners each of which defines a mapping between a load balancer port and an instance port.

A variety of properties used to control behaviors of listeners are defined as system tags, including idle connection timeout, max connections, healthy threshold, unhealthy threshold and so on. Details can be found in *load balancer system tags*.

Inventory

Name	Description	Optional	Choices	Since
uuid	see *Resource Properties*			0.9
name	see *Resource Properties*			0.9
description	see *Resource Properties*	true		0.9
loadBalancerUuid	load balancer uuid			0.9
loadBalancerPort	the frontend port where the incoming traffics visit; it's bond to the VIP of the load balancer		1 ~ 65536	0.9
instancePort	the backend port where the incoming traffics are routed; it's bound to VM nics on the private network		1 ~ 65336	0.9
protocol	see *protocol*		• http • tcp	0.9
vmNicRefs	see *nic reference*			0.9
createDate	see *Resource Properties*			0.9
lastOpDate	see *Resource Properties*			0.9

Protocol

The protocol defines how the load balancer should route incoming traffic. There are two modes: tcp(layer 4) and http(layer 7). When the protocol is *tcp* which is the default mode, the load balancer will work in pure TCP mode; a full-duplex connection will be established between clients and servers. When the protocol is *http*, connections from clients to the load balancer and from the load balancer to your back-end instance are established respectively,

Example


```
{
    "createDate": "Aug 20, 2015 2:54:14 PM",
    "instancePort": 80,
    "lastOpDate": "Aug 20, 2015 2:54:14 PM",
    "loadBalancerPort": 80,
    "loadBalancerUuid": "0188cec6635845e0b2526a8e7e090e2a",
    "name": "80",
    "protocol": "http",
    "uuid": "ba5f192472ab4fc4b36e5af873f0fec5",
    "vmNicRefs": [
        {
            "createDate": "Aug 20, 2015 2:55:49 PM",
            "id": 18,
            "lastOpDate": "Aug 20, 2015 2:55:49 PM",
```

```
            "listenerUuid": "ba5f192472ab4fc4b36e5af873f0fec5",
            "status": "Active",
            "vmNicUuid": "35b8aadef2f847d9836bdf06121e1c29"
        },
        {
            "createDate": "Aug 20, 2015 2:55:49 PM",
            "id": 19,
            "lastOpDate": "Aug 20, 2015 2:55:49 PM",
            "listenerUuid": "ba5f192472ab4fc4b36e5af873f0fec5",
            "status": "Active",
            "vmNicUuid": "df7d40a47cb640a9b40001f2f318989a"
        }
    ]
},
```

2.26.4 Backend VM Nics

Users can add a VM instance to a load balancer by joining its nic to the load balancer's listeners. Once the nic joined, the load balancer routes incoming traffics from the *loadBalancerPort* of the VIP to the *instancePort* of the nic according listeners' balancing algorithm. A nic can join different listeners of different load balancers; it's applications' responsibilities to handle traffics from various load balancers.

The load balancer listener encompasses information of joined VM nics into an inventory called *nic reference*, which has properties as following:

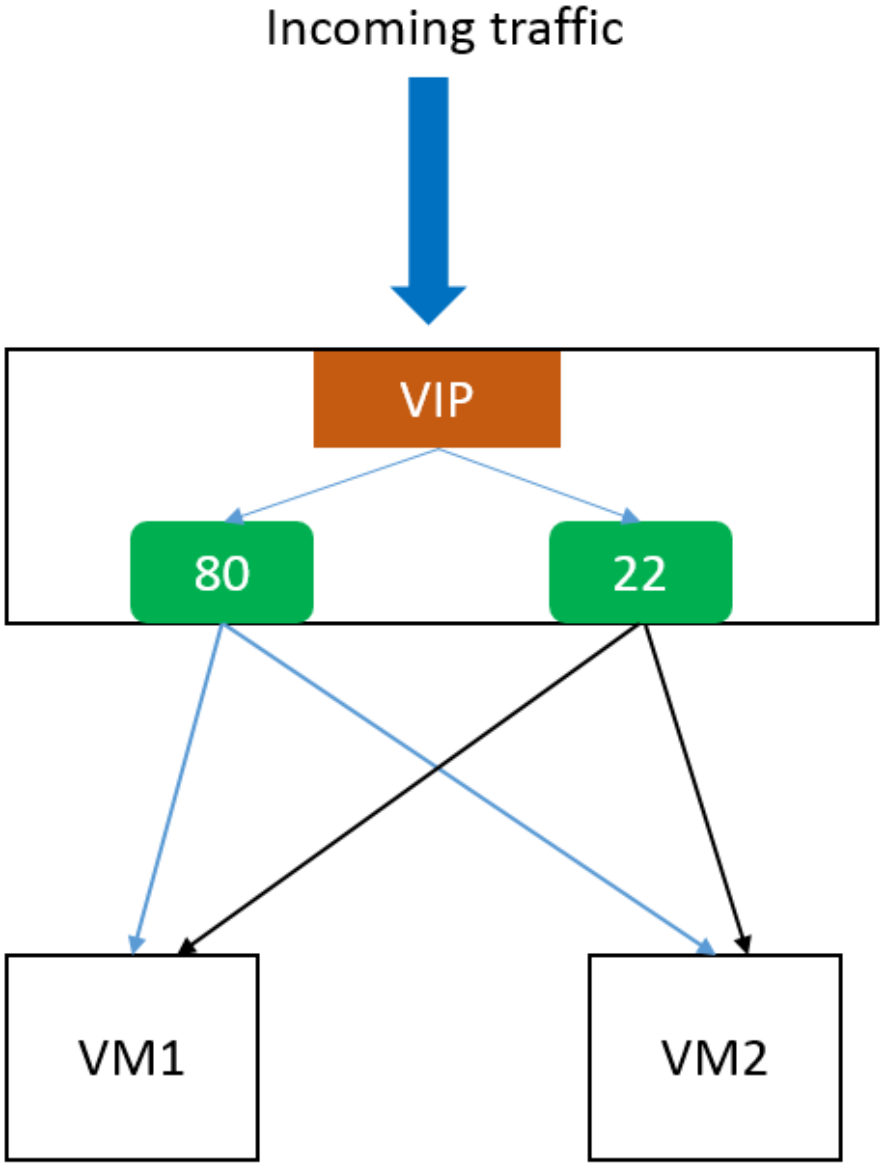
Nic Reference Inventory

Name	Description	Optional	Choices	Since
id	id of the reference			0.9
listenerUuid	listener uuid			0.9
vmNicUuid	VM nic uuid			0.9
status	when the nic's owner VM is running, the status is active; otherwise it's inactive		• Active • Inactive	0.9

After a VM nic joins a load balancer listener, stopping the VM will change the nic status to *Inactive*; starting the VM will change the nic status to *Active*; Destroying the VM will remove the nic from the listener.

2.26.5 A Full Example

Let's say you are about to create a load balancer which routes incoming traffics from port 80 and 22 on the public VIP to two backend VMs.



Public L3 Network UUID	see *Resource Properties*
VM1 nic UUId	35b8aadef2f847d9836bdf06121e1c29
VM2 nic UUID	df7d40a47cb640a9b40001f2f318989a

Create a VIP

:: >>>CreateVip l3NetworkUuid=db6379182e524c06bc8d3ec900ab78d4

Create LB

:: >>>CreateLoadBalancer name=lb vipUuid=df6a73601f1741fd847cf5456b0d42ac

Create listeners

```
CreateLoadBalancerListener loadBalancerUuid=0188cec6635845e0b2526a8e7e090e2a loadBalancerPort=22 inst
```

```
CreateLoadBalancerListener loadBalancerUuid=0188cec6635845e0b2526a8e7e090e2a loadBalancerPort=80 inst
```

Add nics to listeners

```
>>>AddVmNicToLoadBalancer listenerUuid=2901fd13765c492b9a3d004e806a0beb vmNicUuids=35b8aadef2f847d983
```

```
>>>AddVmNicToLoadBalancer listenerUuid=4be2244667d948e286722a4a32e02e65 vmNicUuids=35b8aadef2f847d983
```

2.26.6 Operations

Create Load Balancer

Users can use CreateLoadBalancer to create a load balancer. For example:

```
>>>CreateLoadBalancer name=lb vipUuid=df6a73601f1741fd847cf5456b0d42ac
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.9
resourceUuid	resource uuid, see *Create Resources*	true		0.9
description	resource description, see *Resource Properties*	true		0.9
vipUuid	VIP uuid			0.9
userTags	user tags, see *Create Tags*; resource type is	true		0.9
systemTags	system tags, see *Create Tags*; resource type is	true		0.9

Delete Load Balancer

Users can use DeleteLoadBalancer to delete a load balancer. For example:

```
>>>DeleteLoadBalancer uuid=4be2244667d948e286722a4a32e02e65
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.9
uuid	load balancer uuid			0.9

Create Listener

Users can use CreateLoadBalancerListener to create a load balancer listener. For example:

```
CreateLoadBalancerListener loadBalancerUuid=0188cec6635845e0b2526a8e7e090e2a loadBalancerPort=22 inst
```

Parameters

Name	Description	Optional	Choices	Since
name	resource name, see *Resource Properties*			0.9
resourceUuid	resource uuid, see *Create Resources*	true		0.9
description	resource description, see *Resource Properties*	true		0.9
loadBalancerUuid	load balancer uuid			0.9
loadBalancerPort	frontend load balancer port			0.9
instancePort	backend instance port. If omitted, use loadBalancerPort as instancePort	true		0.9
protocol	see *load balancer protocol*		• tcp • http	0.9
userTags	user tags, see *Create Tags*; resource type is	true		0.9
systemTags	system tags, see *Create Tags*; resource type is	true		0.9

Delete Listener

Users can use DeleteLoadBalancerListener to delete a listener. For example:

```
>>DeleteLoadBalancerListener uuid=0188cec6635845e0b2526a8e7e090e2a
```

Parameters

Name	Description	Optional	Choices	Since
deleteMode	see *Delete Resources*	true	• Permissive • Enforcing	0.9
uuid	listener uuid			0.9

Add VM Nic to Load Balancer

Users can use AddVmNicToLoadBalancer to add VM nics to a load balancer. For example:

```
>>>AddVmNicToLoadBalancer listenerUuid=2901fd13765c492b9a3d004e806a0beb vmNicUuids=35b8aadef2f847d983
```

Parameters

Name	Description	Optional	Choices	Since
listenerUuid	listener uuid			0.9
vmNicUuids	a list of VM nic uuid			0.9

Remove VM Nic from Load Balancer

Users can use RemoveVmNicFromLoadBalancer to remove VM nics from a load balancer. For example:

```
>>>RemoveVmNicFromLoadBalancer listenerUuid=2901fd13765c492b9a3d004e806a0beb vmNicUuids=35b8aadef2f84
```

Parameters

Name	Description	Optional	Choices	Since
listenerUuid	listener uuid			0.9
vmNicUuids	a list of VM nic uuid			0.9

Query Load Balancer

Users can use QueryLoadBalancer to query load balancers. For example:

```
>>>QueryLoadBalancer name=lb
```

```
>>>QueryLoadBalancer listeners.vmNic.vmInstance.name=web
```

Primitive Fields of Query

see *load balancer inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
listeners	see *load balancer listener inventory*	child listeners	0.9
vip	see *vip inventory*	bound VIP	0.9

Query Listener

Users can use QueryLoadBalancerListener to query load balancer listeners. For example:

```
>>>QueryLoadBalancerListener loadBalancerPort=80
```

```
>>>QueryLoadBalancerListener loadBalancer.vip.ip=192.168.0.10
```

Primitive Fields of Query

see *load balancer listener inventory*

Nested And Expanded Fields of Query

Field	Inventory	Description	Since
loadBalancer	see *load balancer inventory*	parent load balancer	0.9
vmNic	see *vm nic inventory*	joined VM nics	0.9

2.26.7 Tags

Users can create user tags on a load balancer with resourceType=LoadBalancerVO. For example:

```
CreateUserTag tag=web-lb resourceUuid=0a9f95a659444848846b5118e15bff32 resourceType=LoadBalancerVO
```

Users can create user tags on a load balancer listener with resourceType=LoadBalancerListenerVO. For example:

```
CreateUserTag tag=web-lb-80 resourceUuid=0a9f95a659444848846b5118e15bff32 resourceType=LoadBalancerLi
```

System Tags

Separate Virtual Router

In this version(0.9), the load balancer service is provided by the virtual router provider. Normally users may need only one virtual router VM providing services like SNAT, EIP, port forwarding and load balancer. However, users can use a system tag to instruct ZStack to spawn an individual virtual router VM for a load balancer. That is to say, creating a virtual router VM dedicated to a load balancer.

Tag	Example	Since
separateVirtualRouterVm	separateVirtualRouterVm	0.9

```
>>>CreateLoadBalancer name=lb vipUuid=df6a73601f1741fd847cf5456b0d42ac systemTags=separateVirtualRout
```

Listener Configurations

A set of system tags can be used to configure a load balancer listener, controlling various listener behaviors such as max connections, idle connection timeout, balancing algorithm and so on. Users can specify those system tags when creating a listener, or ignore them to let ZStack choose default values.

Healthy Threshold The number of consecutive health checks successes required before moving the VM nic to the healthy state.

Tag	Example	Since
healthyThreshold::{healthyThreshold}	healthyThreshold::2	0.9

Health Check Interval The approximate interval, in seconds, between health checks of an individual VM nic

Tag	Example	Since
healthCheckInterval::{healthCheckInterval}	healthCheckInterval::5	0.9

Unhealthy Threshold The number of consecutive health check failures required before moving the instance to the unhealthy state.

Tag	Example	Since
unhealthyThreshold::{unhealthyThreshold}	unhealthyThreshold::2	0.9

Connection Idle Timeout The amount of time, in seconds, during the load balancer closes idle connections on both server and client side.

Tag	Example	Since
connectionIdleTimeout::{connectionIdleTimeout}	60	0.9

Max Connections The max concurrent connections

Tag	Example	Since
maxConnection::{maxConnection}	maxConnection::5000	0.9

Balancing Algorithm The algorithm the load balancer routes incoming traffic; valid choices are: roundrobin, leastconn, source

Tag	Example	Since
balancerAlgorithm::{balancerAlgorithm}	balancerAlgorithm::leastconn	0.9

```
CreateLoadBalancerListener loadBalancerUuid=0188cec6635845e0b2526a8e7e090e2a loadBalancerPort=22 inst
systemTags=maxConnection::10000,balancerAlgorithm::source,healthyThreshold::5
```

2.26.8 Global Configurations

Connection Idle Timeout

The default value of system tag *Connection Idle Timeout*.

Name	Category	Default Value	Choices
connectionIdleTimeout	loadBalancer	60	

Healthy Threshold

The default value of system tag *Healthy Threshold*.

Name	Category	Default Value	Choices
healthyThreshold	loadBalancer	2	

Unhealthy Threshold

The default value of system tag *Unhealthy Threshold*.

Name	Category	Default Value	Choices
unhealthyThreshold	loadBalancer	2	

Health Check Interval

The default value of system tag *Health Check Interval*.

Name	Category	Default Value	Choices
healthCheckInterval	loadBalancer	5	

Max Connection

The default value of system tag *Max Connection*.

Name	Category	Default Value	Choices
maxConnection	loadBalancer	5000	

Balancing Algorithm

The default value of system tag *Balancing Algorithm*.

Name	Category	Default Value	Choices
balancerAlgorithm	loadBalancer	roundrobin	• roundrobin • leastconn • source