
CloudStack Documentation

Version 4.3.0

Apache CloudStack

sept. 18, 2017

Table des matières

1	Introduction	3
1.1	Concepts and Terminology	3
2	Navigating the docs	21
3	Advanced Networking Guides	23
3.1	The Nicira NVP Plugin	23
3.2	The MidoNet Plugin	29
3.3	The VXLAN Plugin	31
3.4	The OVS Plugin	36
3.5	IPv6 Support in CloudStack	44
3.6	Configuring AutoScale without using NetScaler	46
4	Developers Guide	51
4.1	CloudStack Installation from Source for Developers	51
4.2	Programmer Guide	60
4.3	Plugins	76
4.4	Allocators	87
4.5	Deploying CloudStack with Ansible	90



Avertissement : We are in the process of changing documentation format as well as hosting mechanism. Please be patient with us as we migrate our entire documentation to this new setup.

Concepts and Terminology

What is Apache CloudStack ?

Apache CloudStack is an open source Infrastructure-as-a-Service platform that manages and orchestrates pools of storage, network, and computer resources to build a public or private IaaS compute cloud.

With CloudStack you can :

- Set up an on-demand elastic cloud computing service.
- Allow end-users to provision resources

What can Apache CloudStack do ?

Multiple Hypervisor Support

CloudStack works with a variety of hypervisors and hypervisor-like technologies. A single cloud can contain multiple hypervisor implementations. As of the current release CloudStack supports :

- vSphere (via vCenter)
- KVM
- Xenserver
- LXC
- BareMetal (via IPMI)

Massively Scalable Infrastructure Management

CloudStack can manage tens of thousands of physical servers installed in geographically distributed datacenters. The management server scales near-linearly eliminating the need for cluster-level management servers. Maintenance or other outages of the management server can occur without affecting the virtual machines running in the cloud.

Automatic Cloud Configuration Management

CloudStack automatically configures the network and storage settings for each virtual machine deployment. Internally, a pool of virtual appliances support the operation of configuration of the cloud itself. These appliances offer services such as firewalling, routing, DHCP, VPN, console proxy, storage access, and storage replication. The extensive use of horizontally scalable virtual machines simplifies the installation and ongoing operation of a cloud.

Graphical User Interface

CloudStack offers an administrators web interface used for provisioning and managing the cloud, as well as an end-user's Web interface, used for running VMs and managing VM templates. The UI can be customized to reflect the desired service provider or enterprise look and feel.

API

CloudStack provides a REST-like API for the operation, management and use of the cloud.

AWS EC2 API Support

CloudStack provides an EC2 API translation layer to permit the common EC2 tools to be used in the use of a CloudStack cloud.

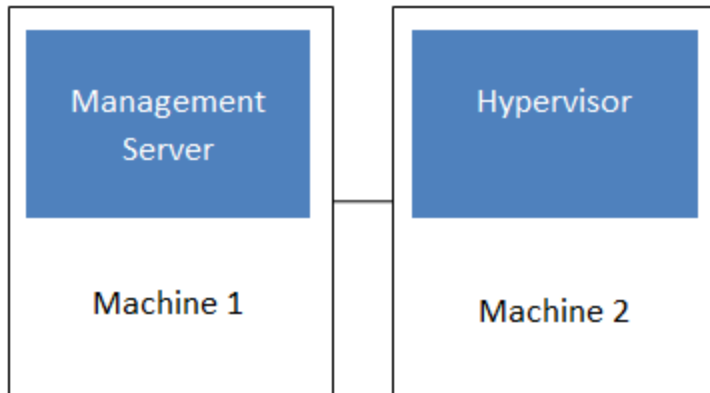
High Availability

CloudStack has a number of features to increase the availability of the system. The Management Server itself may be deployed in a multi-node installation where the servers are load balanced. MySQL may be configured to use replication to provide for failover in the event of database loss. For the hosts, CloudStack supports NIC bonding and the use of separate networks for storage as well as iSCSI Multipath.

Deployment Architecture Overview

Generally speaking, most CloudStack deployments consist of the management server and the resources to be managed. During deployment you inform the management server of the resources to be managed, such as IP address blocks, storage devices, hypervisors, and VLANs.

The minimum installation consists of one machine running the CloudStack Management Server and another machine to act as the cloud infrastructure (in this case, a very simple infrastructure consisting of one host running hypervisor software). In its smallest deployment, a single machine can act as both the Management Server and the hypervisor host (using the KVM hypervisor).



Simplified view of a basic deployment

A more full-featured installation consists of a highly-available multi-node Management Server installation and up to tens of thousands of hosts using any of several networking technologies.

Management Server Overview

The management server orchestrates and allocates the resources in your cloud deployment.

The management server typically runs on a dedicated machine or as a virtual machine. It controls allocation of virtual machines to hosts and assigns storage and IP addresses to the virtual machine instances. The Management Server runs in an Apache Tomcat container and requires a MySQL database for persistence.

The management server :

- Provides the web interface for both the administrator and end user.
- Provides the API interfaces for both the CloudStack API as well as the EC2 interface.
- Manages the assignment of guest VMs to a specific compute resource
- Manages the assignment of public and private IP addresses.
- Allocates storage during the VM instantiation process.
- Manages snapshots, disk images (templates), and ISO images.
- Provides a single point of configuration for your cloud.

Cloud Infrastructure Overview

Resources within the cloud are managed as follows :

- Regions : A collection of one or more geographically proximate zones managed by one or more management servers.
- Zones : Typically, a zone is equivalent to a single datacenter. A zone consists of one or more pods and secondary storage.
- Pods : A pod is usually a rack, or row of racks that includes a layer-2 switch and one or more clusters.
- Clusters : A cluster consists of one or more homogenous hosts and primary storage.
- Host : A single compute node within a cluster ; often a hypervisor.
- Primary Storage : A storage resource typically provided to a single cluster for the actual running of instance disk images. (Zone-wide primary storage is an option, though not typically used.)
- Secondary Storage : A zone-wide resource which stores disk templates, ISO images, and snapshots.

Networking Overview

CloudStack offers many types of networking, but they typically fall into one of two scenarios :

- Basic : Most analogous to AWS-classic style networking. Provides a single flat layer-2 network where guest isolation is provided at layer-3 by the hypervisors bridge device.
- Advanced : This typically uses layer-2 isolation such as VLANs, though this category also includes SDN technologies such as Nicira NVP.

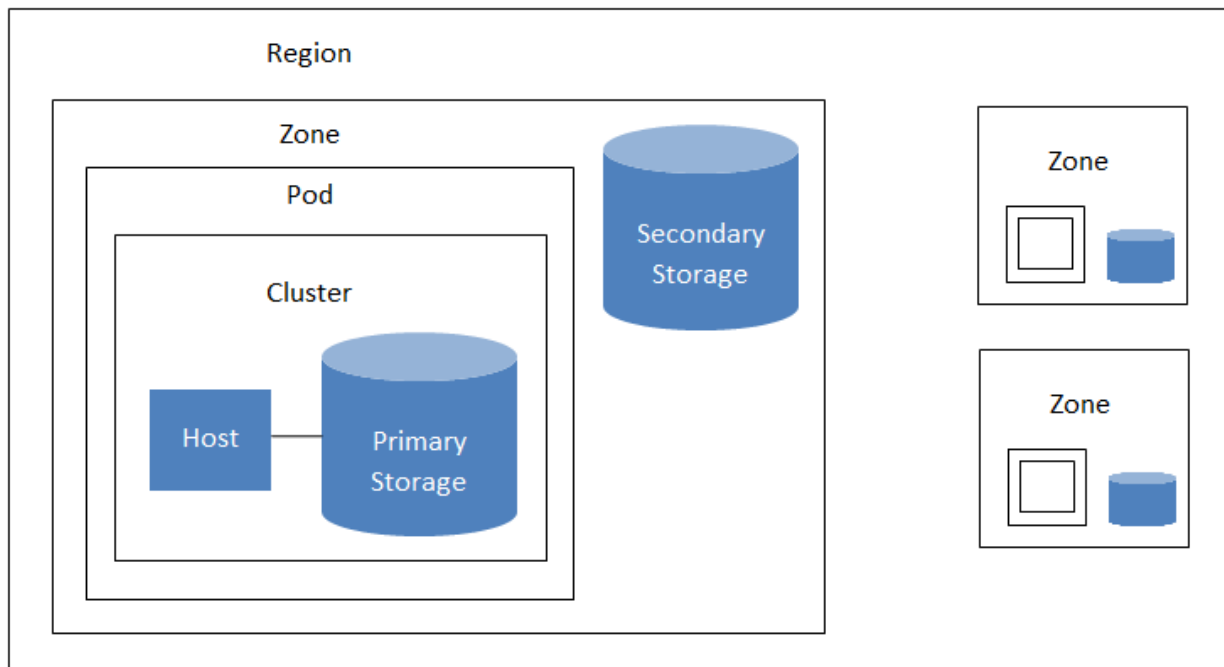
Cloud Infrastructure Concepts

Regions

To increase reliability of the cloud, you can optionally group resources into multiple geographic regions. A region is the largest available organizational unit within a CloudStack deployment. A region is made up of several availability zones, where each zone is roughly equivalent to a datacenter. Each region is controlled by its own cluster of Management Servers, running in one of the zones. The zones in a region are typically located in close geographical proximity. Regions are a useful technique for providing fault tolerance and disaster recovery.

By grouping zones into regions, the cloud can achieve higher availability and scalability. User accounts can span regions, so that users can deploy VMs in multiple, widely-dispersed regions. Even if one of the regions becomes unavailable, the services are still available to the end-user through VMs deployed in another region. And by grouping communities of zones under their own nearby Management Servers, the latency of communications within the cloud is reduced compared to managing widely-dispersed zones from a single central Management Server.

Usage records can also be consolidated and tracked at the region level, creating reports or invoices for each geographic region.



A region with multiple zones

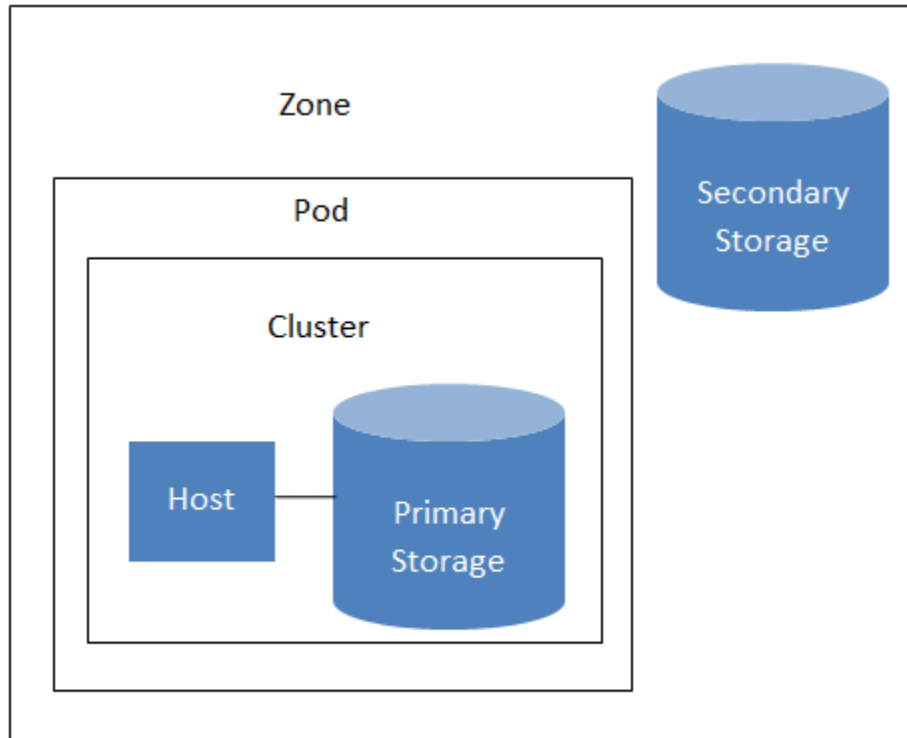
Regions are visible to the end user. When a user starts a guest VM on a particular CloudStack Management Server, the user is implicitly selecting that region for their guest. Users might also be required to copy their private templates to additional regions to enable creation of guest VMs using their templates in those regions.

Zones

A zone is the second largest organizational unit within a CloudStack deployment. A zone typically corresponds to a single datacenter, although it is permissible to have multiple zones in a datacenter. The benefit of organizing infrastructure into zones is to provide physical isolation and redundancy. For example, each zone can have its own power supply and network uplink, and the zones can be widely separated geographically (though this is not required).

A zone consists of :

- One or more pods. Each pod contains one or more clusters of hosts and one or more primary storage servers.
- A zone may contain one or more primary storage servers, which are shared by all the pods in the zone.
- Secondary storage, which is shared by all the pods in the zone.



Nested organization of a zone

Zones are visible to the end user. When a user starts a guest VM, the user must select a zone for their guest. Users might also be required to copy their private templates to additional zones to enable creation of guest VMs using their templates in those zones.

Zones can be public or private. Public zones are visible to all users. This means that any user may create a guest in that zone. Private zones are reserved for a specific domain. Only users in that domain or its subdomains may create guests in that zone.

Hosts in the same zone are directly accessible to each other without having to go through a firewall. Hosts in different zones can access each other through statically configured VPN tunnels.

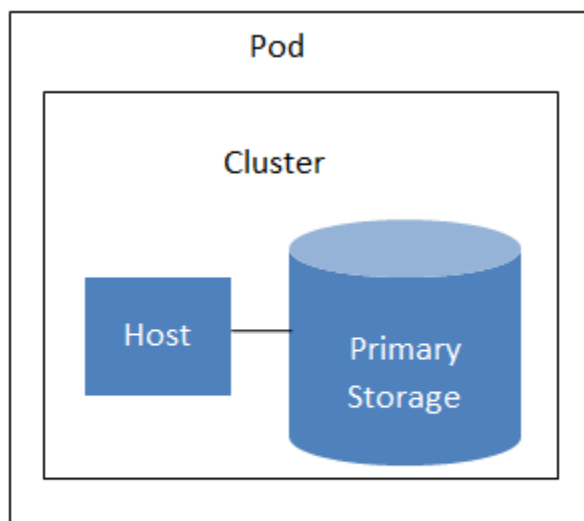
For each zone, the administrator must decide the following.

- How many pods to place in each zone.
- How many clusters to place in each pod.
- How many hosts to place in each cluster.
- (Optional) How many primary storage servers to place in each zone and total capacity for these storage servers.
- How many primary storage servers to place in each cluster and total capacity for these storage servers.
- How much secondary storage to deploy in a zone.

In order to support zone-wide functions for VMware, CloudStack is aware of VMware Datacenters and can map each Datacenter to a CloudStack zone. To enable features like storage live migration and zone-wide primary storage for VMware hosts, CloudStack has to make sure that a zone contains only a single VMware Datacenter. Therefore, when you are creating a new CloudStack zone, you can select a VMware Datacenter for the zone. If you are provisioning multiple VMware Datacenters, each one will be set up as a single zone in CloudStack.

Pods

A pod often represents a single rack or row of racks. Hosts in the same pod are in the same subnet. A pod is the second-largest organizational unit within a CloudStack deployment. Pods are contained within zones. Each zone can contain one or more pods. A pod consists of one or more clusters of hosts and one or more primary storage servers. Pods are not visible to the end user.



A simple pod

Clusters

A cluster consists of one or more hosts and one or more primary storage resources.

A cluster provides a way to group hosts. To be precise, a cluster is a XenServer server pool, a set of KVM servers, or a VMware cluster preconfigured in vCenter. The hosts in a cluster should all have identical hardware, run the same hypervisor, are on the same subnet, and access the same shared primary storage. Virtual machine instances (VMs) can be live-migrated from one host to another within the same cluster, without interrupting service to the user.

The size of the cluster is limited by the underlying hypervisor, although the CloudStack recommends less in most cases ; see Best Practices.

Even when local storage is used exclusively, clusters are still required organizationally, even if there is just one host per cluster.

When VMware is used, every VMware cluster is managed by a vCenter server. An Administrator must register the vCenter server with CloudStack. There may be multiple vCenter servers per zone. Each vCenter server may manage multiple VMware clusters.

Hosts

A host is a single physical computer. Hosts provide the computing resources that run the guest machines.

The host is the smallest organizational unit within a CloudStack deployment and are not visible to an end user.

Primary Storage

Primary storage is associated with a cluster and/or a zone. It stores the disk volumes for all of the VMs running on hosts in that cluster. You can add multiple primary storage servers to a cluster or a zone (at least one is required at the cluster level). Primary storage is typically located close to the hosts for increased performance. CloudStack manages the allocation of guest virtual disks to particular primary storage devices.

Primary storage can be either static or dynamic. Static primary storage is what CloudStack has traditionally supported. In this model, the administrator must present CloudStack with a certain amount of preallocated storage (ex. a volume from a SAN) and CloudStack can place many of its volumes on this storage. In the newer, dynamic model, the administrator can present CloudStack with a storage system itself (i.e. a SAN). CloudStack, working in concert with a plug-in developed for that storage system, can dynamically create volumes on the storage system. A valuable use for this ability is Quality of Service (QoS). If a volume created in CloudStack can be backed by a dedicated volume on a SAN (i.e. a one-to-one mapping between a SAN volume and a CloudStack volume) and the SAN provides QoS functionality, then CloudStack can also orchestrate storage QoS.

CloudStack is designed to work with all standards-compliant iSCSI and NFS servers that are supported by the underlying hypervisor

You may also use local disks as secondary storage, though naturally they don't support live migration.

Secondary Storage

Secondary storage stores the following :

- Templates — OS images that can be used to boot VMs and can include additional configuration information, such as installed applications
- ISO images — disc images containing data or bootable media for operating systems
- Disk volume snapshots — saved copies of VM data which can be used for data recovery or to create new templates

The items in secondary storage are available to all hosts in the scope of the secondary storage, which may be defined as per zone or per region. CloudStack supports both NFS and Object Storage supporting either the AWS S3 API or the Swift API as a backing store for Secondary Storage.

Physical Networks

One or more physical networks can be associated with each zone. The physical network typically corresponds to a physical NIC on the host. Each physical network can carry one or more types of network traffic. The choices of traffic type for each network vary depending on your network choices.

A physical network is the actual network hardware and wiring in a zone. A zone can have multiple physical networks.

- An administrator can :
 - Add/Remove/Update physical networks in a zone
 - Configure VLANs on the physical network
 - Configure a name so the network can be recognized by hypervisors
 - Configure the service providers (firewalls, load balancers, etc.) available on a physical network
 - Configure the IP addresses available to a physical network
 - Specify what type of traffic is carried on the physical network, as well as other properties like network speed

Basic Zone Network Types

When basic networking is used, there can be only one physical network in the zone. That physical network carries the following traffic types :

- **Guest** : When end users run VMs, they generate guest traffic. The guest VMs communicate with each other over a network that can be referred to as the guest network. Each pod in a basic zone is a broadcast domain, and therefore each pod has a different IP range for the guest network. The administrator must configure the IP range for each pod.
- **Management** : When CloudStack's internal resources communicate with each other, they generate management traffic. This includes communication between hosts, system VMs (VMs used by CloudStack to perform various tasks in the cloud), and any other component that communicates directly with the CloudStack Management Server. You must configure the IP range for the system VMs to use.
- **Public** : Public traffic is generated when VMs in the cloud access the Internet. Publicly accessible IPs must be allocated for this purpose. End users can use the CloudStack UI to acquire these IPs to implement NAT between their guest network and the public network, as described in [Acquiring a New IP Address](#).
- **Storage** : While labeled "storage" this is specifically about secondary storage, and doesn't affect traffic for primary storage. This includes traffic such as VM templates and snapshots, which is sent between the secondary storage VM and secondary storage servers. CloudStack uses a separate Network Interface Controller (NIC) named storage NIC for storage network traffic. Use of a storage NIC that always operates on a high bandwidth network allows fast template and snapshot copying. You must configure the IP range to use for the storage network.

In a basic network, configuring the physical network is fairly straightforward. In most cases, you only need to configure one guest network to carry traffic that is generated by guest VMs. If you use a NetScaler load balancer and enable its elastic IP and elastic load balancing (EIP and ELB) features, you must also configure a network to carry public traffic. CloudStack takes care of presenting the necessary network configuration steps to you in the UI when you add a new zone.

Basic Zone Guest IP Addresses

When basic networking is used, CloudStack will assign IP addresses in the CIDR of the pod to the guests in that pod. The administrator must add a Direct IP range on the pod for this purpose. These IPs are in the same VLAN as the hosts.

Advanced Zone Network Types

When advanced networking is used, there can be multiple physical networks in the zone. Each physical network can carry one or more traffic types, and you need to let CloudStack know which type of network traffic you want each network to carry.

The traffic types in an advanced zone are :

- **Guest** : When end users run VMs, they generate guest traffic. The guest VMs communicate with each other over a network that can be referred to as the guest network. This network can be isolated or shared. In an isolated guest network, the administrator needs to reserve VLAN ranges to provide isolation for each CloudStack account's network (potentially a large number of VLANs). In a shared guest network, all guest VMs share a single network.
- **Management** : When CloudStack's internal resources communicate with each other, they generate management traffic. This includes communication between hosts, system VMs (VMs used by CloudStack to perform various tasks in the cloud), and any other component that communicates directly with the CloudStack Management Server. You must configure the IP range for the system VMs to use.
- **Public** : Public traffic is generated when VMs in the cloud access the Internet. Publicly accessible IPs must be allocated for this purpose. End users can use the CloudStack UI to acquire these IPs to implement NAT between their guest network and the public network, as described in ["Acquiring a New IP Address"](#) in the Administration Guide.

- **Storage** : While labeled “storage” this is specifically about secondary storage, and doesn’t affect traffic for primary storage. This includes traffic such as VM templates and snapshots, which is sent between the secondary storage VM and secondary storage servers. CloudStack uses a separate Network Interface Controller (NIC) named storage NIC for storage network traffic. Use of a storage NIC that always operates on a high bandwidth network allows fast template and snapshot copying. You must configure the IP range to use for the storage network.

These traffic types can each be on a separate physical network, or they can be combined with certain restrictions.

Advanced Zone Guest IP Addresses

When advanced networking is used, the administrator can create additional networks for use by the guests. These networks can span the zone and be available to all accounts, or they can be scoped to a single account, in which case only the named account may create guests that attach to these networks. The networks are defined by a VLAN ID, IP range, and gateway. The administrator may provision thousands of these networks if desired. Additionally, the administrator can reserve a part of the IP address space for non-CloudStack VMs and servers.

Advanced Zone Public IP Addresses

In an advanced network, Public IP addresses are typically on one or more dedicated VLANs and are routed or NATED to guest VMs.

System Reserved IP Addresses

In each zone, you need to configure a range of reserved IP addresses for the management network. This network carries communication between the CloudStack Management Server and various system VMs, such as Secondary Storage VMs, Console Proxy VMs, and DHCP.

The reserved IP addresses must be unique across the cloud. You cannot, for example, have a host in one zone which has the same private IP address as a host in another zone.

The hosts in a pod are assigned private IP addresses. These are typically RFC1918 addresses. The Console Proxy and Secondary Storage system VMs are also allocated private IP addresses in the CIDR of the pod that they are created in.

Make sure computing servers and Management Servers use IP addresses outside of the System Reserved IP range. In example, suppose the System Reserved IP range starts at 192.168.154.2 and ends at 192.168.154.7. CloudStack can use .2 to .7 for System VMs. This leaves the rest of the pod CIDR, from .8 to .254, for the Management Server and hypervisor hosts.

In all zones

Provide private IPs for the system in each pod and provision them in CloudStack.

For KVM and XenServer, the recommended number of private IPs per pod is one per host. If you expect a pod to grow, add enough private IPs now to accommodate the growth.

In a zone that uses advanced networking

For zones with advanced networking, we recommend provisioning enough private IPs for your total number of customers, plus enough for the required CloudStack System VMs. Typically, about 10 additional IPs are required for the System VMs. For more information about System VMs, see the section on working with SystemVMs in the Administrator’s Guide.

When advanced networking is being used, the number of private IP addresses available in each pod varies depending on which hypervisor is running on the nodes in that pod. Citrix XenServer and KVM use link-local addresses, which in theory provide more than 65,000 private IP addresses within the address block. As the pod grows over time, this should be more than enough for any reasonable number of hosts as well as IP addresses for guest virtual routers. VMWare ESXi, by contrast uses any administrator-specified subnetting scheme, and the typical administrator provides only 255 IPs per pod. Since these are shared by physical machines, the guest virtual router, and other entities, it is possible to run out of private IPs when scaling up a pod whose nodes are running ESXi.

To ensure adequate headroom to scale private IP space in an ESXi pod that uses advanced networking, use one or both of the following techniques :

- Specify a larger CIDR block for the subnet. A subnet mask with a /20 suffix will provide more than 4,000 IP addresses.
- Create multiple pods, each with its own subnet. In example, if you create 10 pods and each pod has 255 IPs, this will provide 2,550 IP addresses.

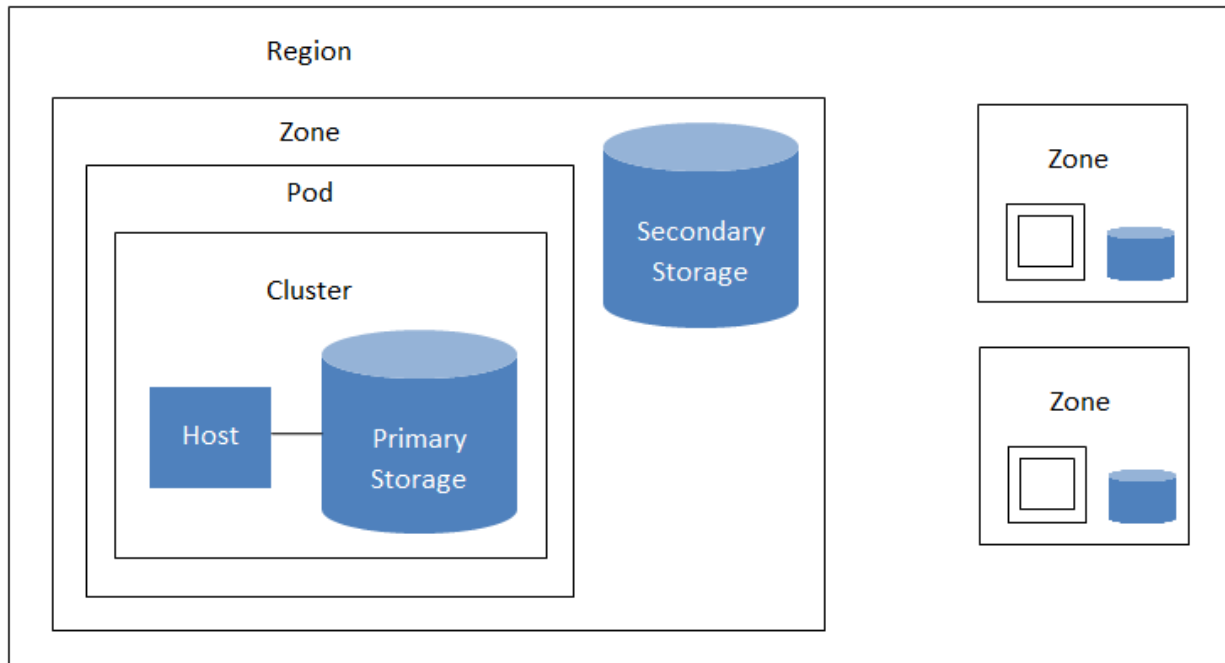
CloudStack Terminology

About Regions

To increase reliability of the cloud, you can optionally group resources into multiple geographic regions. A region is the largest available organizational unit within a CloudStack deployment. A region is made up of several availability zones, where each zone is roughly equivalent to a datacenter. Each region is controlled by its own cluster of Management Servers, running in one of the zones. The zones in a region are typically located in close geographical proximity. Regions are a useful technique for providing fault tolerance and disaster recovery.

By grouping zones into regions, the cloud can achieve higher availability and scalability. User accounts can span regions, so that users can deploy VMs in multiple, widely-dispersed regions. Even if one of the regions becomes unavailable, the services are still available to the end-user through VMs deployed in another region. And by grouping communities of zones under their own nearby Management Servers, the latency of communications within the cloud is reduced compared to managing widely-dispersed zones from a single central Management Server.

Usage records can also be consolidated and tracked at the region level, creating reports or invoices for each geographic region.



A region with multiple zones

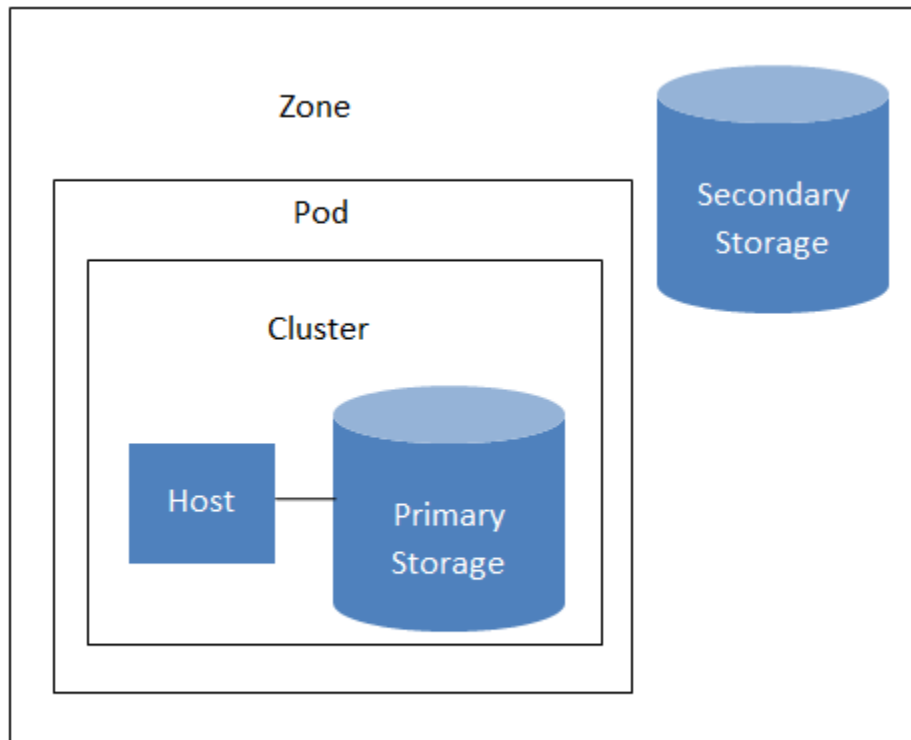
Regions are visible to the end user. When a user starts a guest VM on a particular CloudStack Management Server, the user is implicitly selecting that region for their guest. Users might also be required to copy their private templates to additional regions to enable creation of guest VMs using their templates in those regions.

About Zones

A zone is the second largest organizational unit within a CloudStack deployment. A zone typically corresponds to a single datacenter, although it is permissible to have multiple zones in a datacenter. The benefit of organizing infrastructure into zones is to provide physical isolation and redundancy. For example, each zone can have its own power supply and network uplink, and the zones can be widely separated geographically (though this is not required).

A zone consists of :

- One or more pods. Each pod contains one or more clusters of hosts and one or more primary storage servers.
- A zone may contain one or more primary storage servers, which are shared by all the pods in the zone.
- Secondary storage, which is shared by all the pods in the zone.



Nested organization of a zone

Zones are visible to the end user. When a user starts a guest VM, the user must select a zone for their guest. Users might also be required to copy their private templates to additional zones to enable creation of guest VMs using their templates in those zones.

Zones can be public or private. Public zones are visible to all users. This means that any user may create a guest in that zone. Private zones are reserved for a specific domain. Only users in that domain or its subdomains may create guests in that zone.

Hosts in the same zone are directly accessible to each other without having to go through a firewall. Hosts in different zones can access each other through statically configured VPN tunnels.

For each zone, the administrator must decide the following.

- How many pods to place in each zone.
- How many clusters to place in each pod.
- How many hosts to place in each cluster.
- (Optional) How many primary storage servers to place in each zone and total capacity for these storage servers.
- How many primary storage servers to place in each cluster and total capacity for these storage servers.
- How much secondary storage to deploy in a zone.

When you add a new zone using the CloudStack UI, you will be prompted to configure the zone's physical network and add the first pod, cluster, host, primary storage, and secondary storage.

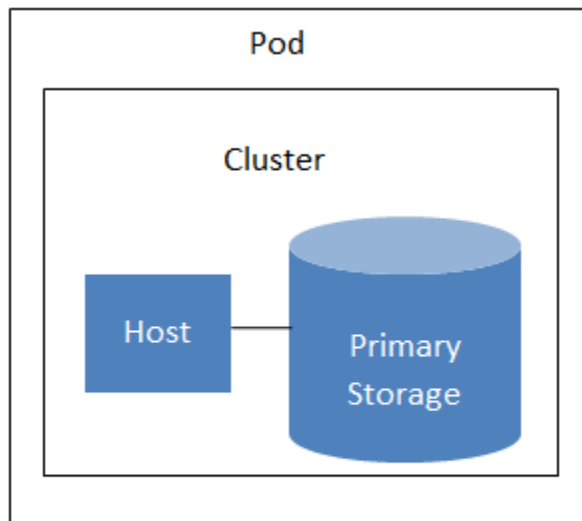
In order to support zone-wide functions for VMware, CloudStack is aware of VMware Datacenters and can map each Datacenter to a CloudStack zone. To enable features like storage live migration and zone-wide primary storage for VMware hosts, CloudStack has to make sure that a zone contains only a single VMware Datacenter. Therefore, when you are creating a new CloudStack zone, you can select a VMware Datacenter for the zone. If you are provisioning multiple VMware Datacenters, each one will be set up as a single zone in CloudStack.

Note : If you are upgrading from a previous CloudStack version, and your existing deployment contains a zone with

clusters from multiple VMware Datacenters, that zone will not be forcibly migrated to the new model. It will continue to function as before. However, any new zone-wide operations, such as zone-wide primary storage and live storage migration, will not be available in that zone.

About Pods

A pod often represents a single rack. Hosts in the same pod are in the same subnet. A pod is the third-largest organizational unit within a CloudStack deployment. Pods are contained within zones. Each zone can contain one or more pods. A pod consists of one or more clusters of hosts and one or more primary storage servers. Pods are not visible to the end user.



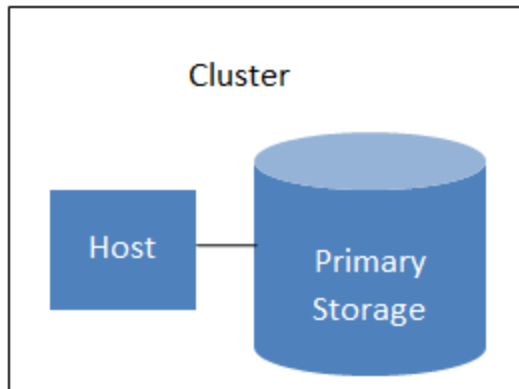
A simple pod

About Clusters

A cluster provides a way to group hosts. To be precise, a cluster is a XenServer server pool, a set of KVM servers, or a VMware cluster preconfigured in vCenter. The hosts in a cluster all have identical hardware, run the same hypervisor, are on the same subnet, and access the same shared primary storage. Virtual machine instances (VMs) can be live-migrated from one host to another within the same cluster, without interrupting service to the user.

A cluster is the fourth-largest organizational unit within a CloudStack deployment. Clusters are contained within pods, and pods are contained within zones. Size of the cluster is limited by the underlying hypervisor, although the CloudStack recommends less in most cases; see Best Practices.

A cluster consists of one or more hosts and one or more primary storage servers.



A simple cluster

CloudStack allows multiple clusters in a cloud deployment.

Even when local storage is used exclusively, clusters are still required organizationally, even if there is just one host per cluster.

When VMware is used, every VMware cluster is managed by a vCenter server. An Administrator must register the vCenter server with CloudStack. There may be multiple vCenter servers per zone. Each vCenter server may manage multiple VMware clusters.

About Hosts

A host is a single computer. Hosts provide the computing resources that run guest virtual machines. Each host has hypervisor software installed on it to manage the guest VMs. For example, a host can be a Citrix XenServer server, a Linux KVM-enabled server, an ESXi server, or a Windows Hyper-V server.

The host is the smallest organizational unit within a CloudStack deployment. Hosts are contained within clusters, clusters are contained within pods, pods are contained within zones, and zones can be contained within regions.

Hosts in a CloudStack deployment :

- Provide the CPU, memory, storage, and networking resources needed to host the virtual machines
- Interconnect using a high bandwidth TCP/IP network and connect to the Internet
- May reside in multiple data centers across different geographic locations
- May have different capacities (different CPU speeds, different amounts of RAM, etc.), although the hosts within a cluster must all be homogeneous

Additional hosts can be added at any time to provide more capacity for guest VMs.

CloudStack automatically detects the amount of CPU and memory resources provided by the hosts.

Hosts are not visible to the end user. An end user cannot determine which host their guest has been assigned to.

For a host to function in CloudStack, you must do the following :

- Install hypervisor software on the host
- Assign an IP address to the host
- Ensure the host is connected to the CloudStack Management Server.

About Primary Storage

Primary storage is associated with a cluster or (in KVM and VMware) a zone, and it stores the disk volumes for all the VMs running on hosts.

You can add multiple primary storage servers to a cluster or zone. At least one is required. It is typically located close to the hosts for increased performance. CloudStack manages the allocation of guest virtual disks to particular primary storage devices.

It is useful to set up zone-wide primary storage when you want to avoid extra data copy operations. With cluster-based primary storage, data in the primary storage is directly available only to VMs within that cluster. If a VM in a different cluster needs some of the data, it must be copied from one cluster to another, using the zone's secondary storage as an intermediate step. This operation can be unnecessarily time-consuming.

For Hyper-V, SMB/CIFS storage is supported. Note that Zone-wide Primary Storage is not supported in Hyper-V.

CloudStack is designed to work with all standards-compliant iSCSI and NFS servers that are supported by the underlying hypervisor, including, for example :

- SolidFire for iSCSI
- Dell EqualLogic™ for iSCSI
- Network Appliances filers for NFS and iSCSI
- Scale Computing for NFS

If you intend to use only local disk for your installation, you can skip adding separate primary storage.

About Secondary Storage

Secondary storage stores the following :

- Templates — OS images that can be used to boot VMs and can include additional configuration information, such as installed applications
- ISO images — disc images containing data or bootable media for operating systems
- Disk volume snapshots — saved copies of VM data which can be used for data recovery or to create new templates

The items in secondary storage are available to all hosts in the scope of the secondary storage, which may be defined as per zone or per region.

To make items in secondary storage available to all hosts throughout the cloud, you can add object storage in addition to the zone-based NFS Secondary Staging Store. It is not necessary to copy templates and snapshots from one zone to another, as would be required when using zone NFS alone. Everything is available everywhere.

For Hyper-V hosts, SMB/CIFS storage is supported.

CloudStack provides plugins that enable both OpenStack Object Storage (Swift, swift.openstack.org) and Amazon Simple Storage Service (S3) object storage. When using one of these storage plugins, you configure Swift or S3 storage for the entire CloudStack, then set up the NFS Secondary Staging Store for each zone. The NFS storage in each zone acts as a staging area through which all templates and other secondary storage data pass before being forwarded to Swift or S3. The backing object storage acts as a cloud-wide resource, making templates and other data available to any zone in the cloud.

Avertissement : Heterogeneous Secondary Storage is not supported in Regions. For example, you cannot set up multiple zones, one using NFS secondary and the other using S3 or Swift secondary.

About Physical Networks

Part of adding a zone is setting up the physical network. One or (in an advanced zone) more physical networks can be associated with each zone. The network corresponds to a NIC on the hypervisor host. Each physical network can carry one or more types of network traffic. The choices of traffic type for each network vary depending on whether you are creating a zone with basic networking or advanced networking.

A physical network is the actual network hardware and wiring in a zone. A zone can have multiple physical networks. An administrator can :

- Add/Remove/Update physical networks in a zone
- Configure VLANs on the physical network
- Configure a name so the network can be recognized by hypervisors
- Configure the service providers (firewalls, load balancers, etc.) available on a physical network
- Configure the IP addresses trunked to a physical network
- Specify what type of traffic is carried on the physical network, as well as other properties like network speed

Basic Zone Network Traffic Types

When basic networking is used, there can be only one physical network in the zone. That physical network carries the following traffic types :

- Guest. When end users run VMs, they generate guest traffic. The guest VMs communicate with each other over a network that can be referred to as the guest network. Each pod in a basic zone is a broadcast domain, and therefore each pod has a different IP range for the guest network. The administrator must configure the IP range for each pod.
- Management. When CloudStack's internal resources communicate with each other, they generate management traffic. This includes communication between hosts, system VMs (VMs used by CloudStack to perform various tasks in the cloud), and any other component that communicates directly with the CloudStack Management Server. You must configure the IP range for the system VMs to use.

Note : We strongly recommend the use of separate NICs for management traffic and guest traffic.

- Public. Public traffic is generated when VMs in the cloud access the Internet. Publicly accessible IPs must be allocated for this purpose. End users can use the CloudStack UI to acquire these IPs to implement NAT between their guest network and the public network, as described in [Acquiring a New IP Address](#).
- Storage. While labeled "storage" this is specifically about secondary storage, and doesn't affect traffic for primary storage. This includes traffic such as VM templates and snapshots, which is sent between the secondary storage VM and secondary storage servers. CloudStack uses a separate Network Interface Controller (NIC) named storage NIC for storage network traffic. Use of a storage NIC that always operates on a high bandwidth network allows fast template and snapshot copying. You must configure the IP range to use for the storage network.

In a basic network, configuring the physical network is fairly straightforward. In most cases, you only need to configure one guest network to carry traffic that is generated by guest VMs. If you use a NetScaler load balancer and enable its elastic IP and elastic load balancing (EIP and ELB) features, you must also configure a network to carry public traffic. CloudStack takes care of presenting the necessary network configuration steps to you in the UI when you add a new zone.

Basic Zone Guest IP Addresses

When basic networking is used, CloudStack will assign IP addresses in the CIDR of the pod to the guests in that pod. The administrator must add a Direct IP range on the pod for this purpose. These IPs are in the same VLAN as the hosts.

Advanced Zone Network Traffic Types

When advanced networking is used, there can be multiple physical networks in the zone. Each physical network can carry one or more traffic types, and you need to let CloudStack know which type of network traffic you want each network to carry. The traffic types in an advanced zone are :

- Guest. When end users run VMs, they generate guest traffic. The guest VMs communicate with each other over a network that can be referred to as the guest network. This network can be isolated or shared. In an isolated

guest network, the administrator needs to reserve VLAN ranges to provide isolation for each CloudStack account's network (potentially a large number of VLANs). In a shared guest network, all guest VMs share a single network.

- **Management.** When CloudStack's internal resources communicate with each other, they generate management traffic. This includes communication between hosts, system VMs (VMs used by CloudStack to perform various tasks in the cloud), and any other component that communicates directly with the CloudStack Management Server. You must configure the IP range for the system VMs to use.
- **Public.** Public traffic is generated when VMs in the cloud access the Internet. Publicly accessible IPs must be allocated for this purpose. End users can use the CloudStack UI to acquire these IPs to implement NAT between their guest network and the public network, as described in "Acquiring a New IP Address" in the Administration Guide.
- **Storage.** While labeled "storage" this is specifically about secondary storage, and doesn't affect traffic for primary storage. This includes traffic such as VM templates and snapshots, which is sent between the secondary storage VM and secondary storage servers. CloudStack uses a separate Network Interface Controller (NIC) named storage NIC for storage network traffic. Use of a storage NIC that always operates on a high bandwidth network allows fast template and snapshot copying. You must configure the IP range to use for the storage network.

These traffic types can each be on a separate physical network, or they can be combined with certain restrictions. When you use the Add Zone wizard in the UI to create a new zone, you are guided into making only valid choices.

Advanced Zone Guest IP Addresses

When advanced networking is used, the administrator can create additional networks for use by the guests. These networks can span the zone and be available to all accounts, or they can be scoped to a single account, in which case only the named account may create guests that attach to these networks. The networks are defined by a VLAN ID, IP range, and gateway. The administrator may provision thousands of these networks if desired. Additionally, the administrator can reserve a part of the IP address space for non-CloudStack VMs and servers.

Advanced Zone Public IP Addresses

When advanced networking is used, the administrator can create additional networks for use by the guests. These networks can span the zone and be available to all accounts, or they can be scoped to a single account, in which case only the named account may create guests that attach to these networks. The networks are defined by a VLAN ID, IP range, and gateway. The administrator may provision thousands of these networks if desired.

System Reserved IP Addresses

In each zone, you need to configure a range of reserved IP addresses for the management network. This network carries communication between the CloudStack Management Server and various system VMs, such as Secondary Storage VMs, Console Proxy VMs, and DHCP.

The reserved IP addresses must be unique across the cloud. You cannot, for example, have a host in one zone which has the same private IP address as a host in another zone.

The hosts in a pod are assigned private IP addresses. These are typically RFC1918 addresses. The Console Proxy and Secondary Storage system VMs are also allocated private IP addresses in the CIDR of the pod that they are created in.

Make sure computing servers and Management Servers use IP addresses outside of the System Reserved IP range. For example, suppose the System Reserved IP range starts at 192.168.154.2 and ends at 192.168.154.7. CloudStack can use .2 to .7 for System VMs. This leaves the rest of the pod CIDR, from .8 to .254, for the Management Server and hypervisor hosts.

In all zones :

Provide private IPs for the system in each pod and provision them in CloudStack.

For KVM and XenServer, the recommended number of private IPs per pod is one per host. If you expect a pod to grow, add enough private IPs now to accommodate the growth.

In a zone that uses advanced networking :

For zones with advanced networking, we recommend provisioning enough private IPs for your total number of customers, plus enough for the required CloudStack System VMs. Typically, about 10 additional IPs are required for the System VMs. For more information about System VMs, see the section on working with SystemVMs in the Administrator's Guide.

When advanced networking is being used, the number of private IP addresses available in each pod varies depending on which hypervisor is running on the nodes in that pod. Citrix XenServer and KVM use link-local addresses, which in theory provide more than 65,000 private IP addresses within the address block. As the pod grows over time, this should be more than enough for any reasonable number of hosts as well as IP addresses for guest virtual routers. VMWare ESXi, by contrast uses any administrator-specified subnetting scheme, and the typical administrator provides only 255 IPs per pod. Since these are shared by physical machines, the guest virtual router, and other entities, it is possible to run out of private IPs when scaling up a pod whose nodes are running ESXi.

To ensure adequate headroom to scale private IP space in an ESXi pod that uses advanced networking, use one or both of the following techniques :

- Specify a larger CIDR block for the subnet. A subnet mask with a /20 suffix will provide more than 4,000 IP addresses.
- Create multiple pods, each with its own subnet. For example, if you create 10 pods and each pod has 255 IPs, this will provide 2,550 IP addresses.

CHAPITRE 2

Navigating the docs

Now that you have gone over the basic concepts of CloudStack you are ready to dive into the installation and operation.

See the [Installation Guide](#)

See the [Administration Guide](#)

See the [Release Notes](#)

Below you will find very specific documentation on advanced *networking* which you can skip if you are just getting started.

Developers will also find below a short *developers* guide.

The Nicira NVP Plugin

Introduction to the Nicira NVP Plugin

The Nicira NVP plugin adds Nicira NVP as one of the available SDN implementations in CloudStack. With the plugin an existing Nicira NVP setup can be used by CloudStack to implement isolated guest networks and to provide additional services like routing and NAT.

Features of the Nicira NVP Plugin

The following table lists the CloudStack network services provided by the Nicira NVP Plugin.

Network Service	CloudStack version	NVP version
Virtual Networking	>= 4.0	>= 2.2.1
Source NAT	>= 4.1	>= 3.0.1
Static NAT	>= 4.1	>= 3.0.1
Port Forwarding	>= 4.1	>= 3.0.1

Table : Supported Services

Note : The Virtual Networking service was originally called ‘Connectivity’ in CloudStack 4.0

The following hypervisors are supported by the Nicira NVP Plugin.

Hypervisor	CloudStack version
XenServer	>= 4.0
KVM	>= 4.1

Table : Supported Hypervisors

Note : Please refer to the Nicira NVP configuration guide on how to prepare the hypervisors for Nicira NVP integration.

Configuring the Nicira NVP Plugin

Prerequisites

Before enabling the Nicira NVP plugin the NVP Controller needs to be configured. Please review the NVP User Guide on how to do that.

Make sure you have the following information ready :

- The IP address of the NVP Controller
- The username to access the API
- The password to access the API
- The UUID of the Transport Zone that contains the hypervisors in this Zone
- The UUID of the Gateway Service used to provide router and NAT services.

Note : The gateway service uuid is optional and is used for Layer 3 services only (SourceNat, StaticNat and PortForwarding)

Zone Configuration

CloudStack needs to have at least one physical network with the isolation method set to “STT”. This network should be enabled for the Guest traffic type.

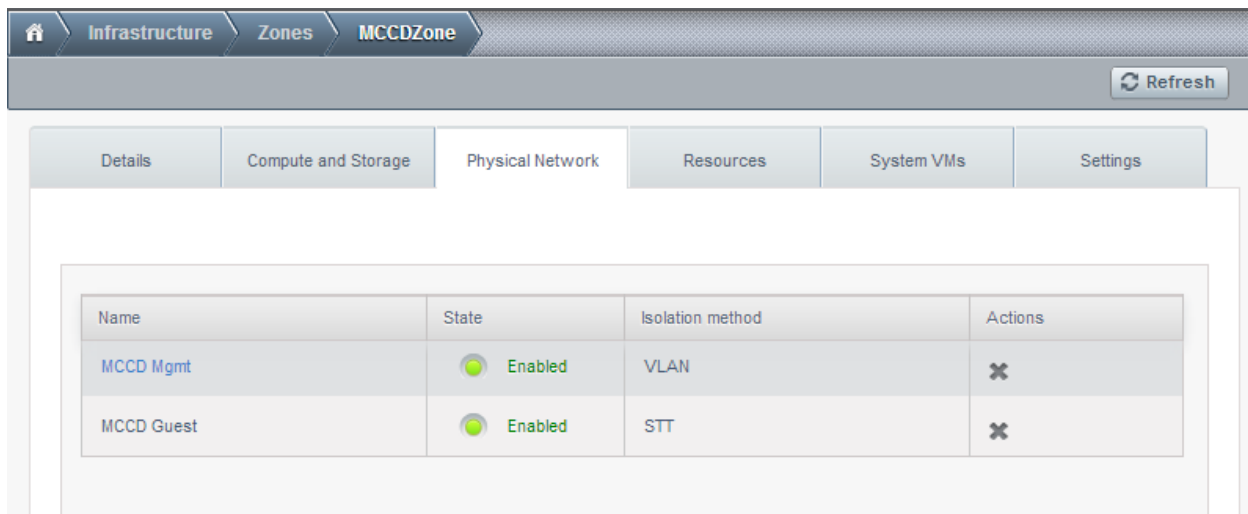
Note : The Guest traffic type should be configured with the traffic label that matches the name of the Integration Bridge on the hypervisor. See the Nicira NVP User Guide for more details on how to set this up in XenServer or KVM.

Name	State	Isolation method	Actions
MCD Mgmt	Enabled	VLAN	X
MCD Guest	Enabled	STT	X

Enabling the service provider

The Nicira NVP provider is disabled by default. Navigate to the “Network Service Providers” configuration of the physical network with the STT isolation type. Navigate to the Nicira NVP provider and press the “Enable Provider” button.

Note : CloudStack 4.0 does not have the UI interface to configure the Nicira NVP plugin. Configuration needs to be done using the API directly.

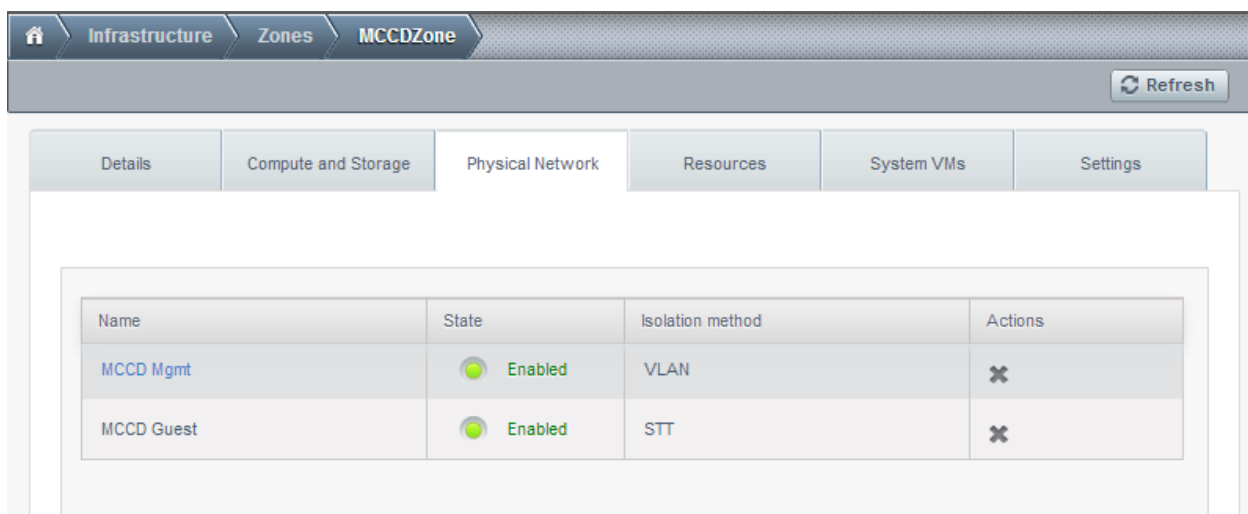


The screenshot shows the CloudStack UI with the breadcrumb navigation: Infrastructure > Zones > MCCDZone. A 'Refresh' button is in the top right. Below the navigation tabs (Details, Compute and Storage, Physical Network, Resources, System VMs, Settings), a table displays the network providers:

Name	State	Isolation method	Actions
MCCD Mgmt	Enabled	VLAN	✕
MCCD Guest	Enabled	STT	✕

Device Management

In CloudStack a Nicira NVP setup is considered a “device” that can be added and removed from a physical network. To complete the configuration of the Nicira NVP plugin a device needs to be added to the physical network. Press the “Add NVP Controller” button on the provider panel and enter the configuration details.



This screenshot is identical to the one above, showing the same UI elements and table data for the 'Physical Network' tab in the 'MCCDZone'.

Network Offerings

Using the Nicira NVP plugin requires a network offering with Virtual Networking enabled and configured to use the NiciraNvp element. Typical use cases combine services from the Virtual Router appliance and the Nicira NVP plugin.

Service	Provider
VPN	VirtualRouter
DHCP	VirtualRouter
DNS	VirtualRouter
Firewall	VirtualRouter
Load Balancer	VirtualRouter
User Data	VirtualRouter
Source NAT	VirtualRouter
Static NAT	VirtualRouter
Post Forwarding	VirtualRouter
Virtual Networking	NiciraNVP

Table : Isolated network offering with regular services from the Virtual Router.

The screenshot shows the CloudStack web interface. The breadcrumb navigation at the top indicates the path: Infrastructure > Zones > MCCDZone. Below the navigation bar, there are tabs for 'Details', 'Compute and Storage', 'Physical Network', 'Resources', 'System VMs', and 'Settings'. The 'Physical Network' tab is selected. Inside this tab, there is a table with the following data:

Name	State	Isolation method	Actions
MCCD Mgmt	Enabled	VLAN	X
MCCD Guest	Enabled	STT	X

Note : The tag in the network offering should be set to the name of the physical network with the NVP provider.

Isolated network with network services. The virtual router is still required to provide network services like dns and dhcp.

Service	Provider
DHCP	VirtualRouter
DNS	VirtualRouter
User Data	VirtualRouter
Source NAT	NiciraNVP
Static NAT	NiciraNVP
Post Forwarding	NiciraNVP
Virtual Networking	NiciraNVP

Table : Isolated network offering with network services

Using the Nicira NVP plugin with VPC

Supported VPC features

The Nicira NVP plugin supports CloudStack VPC to a certain extent. Starting with CloudStack version 4.1 VPCs can be deployed using NVP isolated networks.

It is not possible to use a Nicira NVP Logical Router for as a VPC Router

It is not possible to connect a private gateway using a Nicira NVP Logical Switch

VPC Offering with Nicira NVP

To allow a VPC to use the Nicira NVP plugin to provision networks, a new VPC offering needs to be created which allows the Virtual Networking service to be implemented by NiciraNVP.

This is not currently possible with the UI. The API does provide the proper calls to create a VPC offering with Virtual Networking enabled. However due to a limitation in the 4.1 API it is not possible to select the provider for this network service. To configure the VPC offering with the NiciraNVP provider edit the database table 'vpc_offering_service_map' and change the provider to NiciraNvp for the service 'Connectivity'

It is also possible to update the default VPC offering by adding a row to the 'vpc_offering_service_map' with service 'Connectivity' and provider 'NiciraNvp'

Name	State	Isolation method	Actions
MCCD Mgmt	Enabled	VLAN	✕
MCCD Guest	Enabled	STT	✕

Note : When creating a new VPC offering please note that the UI does not allow you to select a VPC offering yet. The VPC needs to be created using the API with the offering UUID.

VPC Network Offerings

The VPC needs specific network offerings with the VPC flag enabled. Otherwise these network offerings are identical to regular network offerings. To allow VPC networks with a Nicira NVP isolated network the offerings need to support the Virtual Networking service with the NiciraNVP provider.

In a typical configuration two network offerings need to be created. One with the loadbalancing service enabled and one without loadbalancing.

Service	Provider
VPN	VpcVirtualRouter
DHCP	VpcVirtualRouter
DNS	VpcVirtualRouter
Load Balancer	VpcVirtualRouter
User Data	VpcVirtualRouter
Source NAT	VpcVirtualRouter
Static NAT	VpcVirtualRouter
Port Forwarding	VpcVirtualRouter
NetworkACL	VpcVirtualRouter
Virtual Networking	NiciraNVP

Table : VPC Network Offering with Loadbalancing

Troubleshooting the Nicira NVP Plugin

UUID References

The plugin maintains several references in the CloudStack database to items created on the NVP Controller.

Every guest network that is created will have its broadcast type set to Lswitch and if the network is in state “Implemented”, the broadcast URI will have the UUID of the Logical Switch that was created for this network on the NVP Controller.

The Nics that are connected to one of the Logical Switches will have their Logical Switch Port UUID listed in the `nicira_nvp_nic_map` table

Note : All devices created on the NVP Controller will have a tag set to domain-account of the owner of the network, this string can be used to search for items in the NVP Controller.

Database tables

The following tables are added to the cloud database for the Nicira NVP Plugin

id	auto incrementing id
logicalswitch	uuid of the logical switch this port is connected to
logicalswitchport	uuid of the logical switch port for this nic
nic	the CloudStack uuid for this nic, reference to the nics table

Table : `nicira_nvp_nic_map`

id	auto incrementing id
uuid	UUID identifying this device
physical_network_id	the physical network this device is configured on
provider_name	NiciraNVP
device_name	display name for this device
host_id	reference to the host table with the device configuration

Table : `external_nicira_nvp_devices`

id	auto incrementing id
logicalrouter_uuid	uuid of the logical router
network_id	id of the network this router is linked to

Table : `nicira_nvp_router_map`

Note : nicira_nvp_router_map is only available in CloudStack 4.1 and above

Revision History

0-0 Wed Oct 03 2012 Hugo Trippaers hugo@apache.org Documentation created for 4.0.0-incubating version of the NVP Plugin
1-0 Wed May 22 2013 Hugo Trippaers hugo@apache.org Documentation updated for CloudStack 4.1.0

The MidoNet Plugin

Introduction to the MidoNet Plugin

The MidoNet plugin allows CloudStack to use the MidoNet virtualized networking solution as a provider for CloudStack networks and services. For more information on MidoNet and how it works, see <http://www.midokura.com/midonet/>.

Features of the MidoNet Plugin

Note : In CloudStack 4.2.0 only the KVM hypervisor is supported for use in combination with MidoNet.

In CloudStack release 4.2.0 this plugin supports several services in the Advanced Isolated network mode.

When tenants create new isolated layer 3 networks, instead of spinning up extra Virtual Router VMs, the relevant L3 elements (routers etc) are created in the MidoNet virtual topology by making the appropriate calls to the MidoNet API. Instead of using VLANs, isolation is provided by MidoNet.

Aside from the above service (Connectivity), several extra features are supported in the 4.2.0 release :

- DHCP
- Firewall (ingress)
- Source NAT
- Static NAT
- Port Forwarding

The plugin has been tested with MidoNet version 12.12. (Caddo).

Using the MidoNet Plugin

Prerequisites

In order to use the MidoNet plugin, the compute hosts must be running the MidoNet Agent, and the MidoNet API server must be available. Please consult the MidoNet User Guide for more information. The following section describes the CloudStack side setup.

1. CloudStack needs to have at least one physical network with the isolation method set to “MIDO”. This network should be enabled for the Guest and Public traffic types.
2. Next, we need to set the following CloudStack settings under “Global Settings” in the UI :

Setting Name	Description	Example
mido-net.apiserver.address	Specify the address at which the Midonet API server can be contacted	http://192.168.1.144:8081/midolmanj-mgmt
mido-net.providerrouter.id	Specifies the UUID of the Midonet provider router	d7c5e6a3-e2f4-426b-b728-b7ce6a0448e5

Table : CloudStack settings

3. We also want MidoNet to take care of public traffic, so in *componentContext.xml* we need to replace this line :

```
<bean id="PublicNetworkGuru" class="com.cloud.network.guru.PublicNetworkGuru">
```

With this :

```
<bean id="PublicNetworkGuru" class="com.cloud.network.guru.  
↳MidoNetPublicNetworkGuru">
```

Note : On the compute host, MidoNet takes advantage of per-traffic type VIF driver support in CloudStack KVM.

In *agent.properties*, we set the following to make MidoNet take care of Guest and Public traffic :

```
libvirt.vif.driver.Guest=com.cloud.network.resource.MidoNetVifDriver  
libvirt.vif.driver.Public=com.cloud.network.resource.MidoNetVifDriver
```

This is explained further in MidoNet User Guide.

Enabling the MidoNet service provider via the UI

To allow CloudStack to use the MidoNet Plugin the network service provider needs to be enabled on the physical network.

The steps to enable via the UI are as follows :

1. In the left navbar, click Infrastructure
2. In Zones, click View All
3. Click the name of the Zone on which you are setting up MidoNet
4. Click the Physical Network tab
5. Click the Name of the Network on which you are setting up MidoNet
6. Click Configure on the Network Service Providers box
7. Click on the name MidoNet
8. Click the Enable Provider button in the Network tab

Enabling the MidoNet service provider via the API

To enable via the API, use the following API calls :

addNetworkServiceProvider

- name = "MidoNet"
- physicalnetworkid = <the uuid of the physical network>

updateNetworkServiceProvider

- id = <the provider uuid returned by the previous call>
- state = "Enabled"

Revision History

0-0 Wed Mar 13 2013 Dave Cahill dcahill@midokura.com Documentation created for 4.2.0 version of the MidoNet Plugin

The VXLAN Plugin

System Requirements for VXLAN

In PRODUCT 4.X.0, this plugin only supports the KVM hypervisor with the standard linux bridge.

The following table lists the requirements for the hypervisor.

Item	Requirement	Note
Hyper-visor	KVM	OvsVifDriver is not supported by this plugin in PRODUCT 4.X, use BridgeVifDriver (default).
Linux kernel	version \geq 3.7, VXLAN kernel module enabled	It is recommended to use kernel \geq 3.9, since Linux kernel categorizes the VXLAN driver as experimental $<$ 3.9.
iproute2	matches kernel version	

Table : Hypervisor Requirement for VXLAN

Linux Distributions that meet the requirements

The following table lists distributions which meet requirements.

Distribu-tion	Release Version	Kernel Version (Date confirmed)	Note
Ubuntu	13.04	3.8.0 (2013/07/23)	
Fedora	\geq 17	3.9.10 (2013/07/23)	Latest kernel packages are available in "update" repository.
CentOS	\geq 6.5	2.6.32-431.3.1.el6.x86_64 (2014/01/21)	

Table : List of Linux distributions which meet the hypervisor requirements

Check the capability of your system

To check the capability of your system, execute the following commands.

```
$ sudo modprobe vxlan && echo $?
# Confirm the output is "0".
# If it's non-0 value or error message, your kernel doesn't have VXLAN kernel module.

$ ip link add type vxlan help
# Confirm the output is usage of the command and that it's for VXLAN.
# If it's not, your iproute2 utility doesn't support VXLAN.
```

Advanced : Build kernel and iproute2

Even if your system doesn't support VXLAN, you can compile the kernel and iproute2 by yourself. The following procedure is an example for CentOS 6.4.

Build kernel

```
$ sudo yum groupinstall "Development Tools"
$ sudo yum install ncurses-devel hmaccalc zlib-devel binutils-devel elfutils-libelf-
→devel bc

$ KERNEL_VERSION=3.10.4
# Declare the kernel version you want to build.

$ wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-${KERNEL_VERSION}.tar.xz
$ tar xvf linux-${KERNEL_VERSION}.tar.xz
$ cd linux-${KERNEL_VERSION}
$ cp /boot/config-`uname -r` .config
$ make oldconfig
# You may keep hitting enter and choose the default.

$ make menuconfig
# Dig into "Device Drivers" -> "Network device support",
# then select "Virtual eXtensible Local Area Network (VXLAN)" and hit space.
# Make sure it indicates "<M>" (build as module), then Save and Exit.

# You may also want to check "IPv4 NAT" and its child nodes in "IP: Netfilter_
→Configuration"
# and "IPv6 NAT" and its child nodes in "IPv6: Netfilter Configuration".
# In 3.10.4, you can find the options in
# "Networking support" -> "Networking options"
# -> "Network packet filtering framework (Netfilter)".

$ make # -j N
# You may use -j N option to make the build process parallel and faster,
# generally N = 1 + (cores your machine have).

$ sudo make modules_install
$ sudo make install
# You would get an error like "ERROR: modinfo: could not find module XXXX" here.
# This happens mainly due to config structure changes between kernel versions.
# You can ignore this error, until you find you need the kernel module.
# If you feel uneasy, you can go back to make menuconfig,
# find module XXXX by using '/' key, enable the module, build and install the kernel_
→again.

$ sudo vi /etc/grub.conf
# Make sure the new kernel isn't set as the default and the timeout is long enough,
# so you can select the new kernel during boot process.
# It's not a good idea to set the new kernel as the default until you confirm the_
→kernel works fine.

$ sudo reboot
# Select the new kernel during the boot process.
```

Build iproute2

```
$ sudo yum install db4-devel

$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/shemminger/iproute2.git
$ cd iproute2
```

```
$ git tag
# Find the version that matches the kernel.
# If you built kernel 3.10.4 as above, it would be v3.10.0.

$ git checkout v3.10.0
$ ./configure
$ make # -j N
$ sudo make install
```

Note : Please use rebuild kernel and tools at your own risk.

Configure PRODUCT to use VXLAN Plugin

Configure hypervisor

Configure hypervisor : KVM

In addition to “KVM Hypervisor Host Installation” in “PRODUCT Installation Guide”, you have to configure the following item on the host.

Create bridge interface with IPv4 address

This plugin requires an IPv4 address on the KVM host to terminate and originate VXLAN traffic. The address should be assigned to a physical interface or a bridge interface bound to a physical interface. Both a private address or a public address are fine for the purpose. It is not required to be in the same subnet for all hypervisors in a zone, but they should be able to reach each other via IP multicast with UDP/8472 port. A name of a physical interface or a name of a bridge interface bound to a physical interface can be used as a traffic label. Physical interface name fits for almost all cases, but if physical interface name differs per host, you may use a bridge to set a same name. If you would like to use a bridge name as a traffic label, you may create a bridge in this way.

Let `cloudbr1` be the bridge interface for the instances’ private network.

Configure in RHEL or CentOS

When you configured the `cloudbr1` interface as below,

```
$ sudo vi /etc/sysconfig/network-scripts/ifcfg-cloudbr1
```

```
DEVICE=cloudbr1
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=none
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
STP=yes
```

you would change the configuration similar to below.

```
DEVICE=cloudbr1
TYPE=Bridge
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.0.2.X
NETMASK=255.255.255.0
IPV6INIT=no
IPV6_AUTOCONF=no
DELAY=5
STP=yes
```

Configure in Ubuntu

When you configured cloudbr1 as below,

```
$ sudo vi /etc/network/interfaces
```

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0.100
iface eth0.100 inet static
    address 192.168.42.11
    netmask 255.255.255.240
    gateway 192.168.42.1
    dns-nameservers 8.8.8.8 8.8.4.4
    dns-domain lab.example.org

# Public network
auto cloudbr0
iface cloudbr0 inet manual
    bridge_ports eth0.200
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1

# Private network
auto cloudbr1
iface cloudbr1 inet manual
    bridge_ports eth0.300
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1
```

you would change the configuration similar to below.

```
auto lo
iface lo inet loopback

# The primary network interface
auto eth0.100
iface eth0.100 inet static
    address 192.168.42.11
    netmask 255.255.255.240
    gateway 192.168.42.1
```

```

dns-nameservers 8.8.8.8 8.8.4.4
dns-domain lab.example.org

# Public network
auto cloudbri0
iface cloudbri0 inet manual
    bridge_ports eth0.200
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1

# Private network
auto cloudbri1
iface cloudbri1 inet static
    address 192.0.2.X
    netmask 255.255.255.0
    bridge_ports eth0.300
    bridge_fd 5
    bridge_stp off
    bridge_maxwait 1

```

Configure iptables to pass VXLAN packets

Since VXLAN uses UDP packet to forward encapsulated the L2 frames, UDP/8472 port must be opened.

Configure in RHEL or CentOS

RHEL and CentOS use iptables for firewalling the system, you can open extra ports by executing the following iptable commands :

```
$ sudo iptables -I INPUT -p udp -m udp --dport 8472 -j ACCEPT
```

These iptable settings are not persistent accross reboots, we have to save them first.

```
$ sudo iptables-save > /etc/sysconfig/iptables
```

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

```
$ sudo service network restart
$ sudo reboot
```

Avertissement : Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning !

Configure in Ubuntu

The default firewall under Ubuntu is UFW (Uncomplicated FireWall), which is a Python wrapper around iptables.

To open the required ports, execute the following commands :

```
$ sudo ufw allow proto udp from any to any port 8472
```

Note : By default UFW is not enabled on Ubuntu. Executing these commands with the firewall disabled does not enable the firewall.

With this configuration you should be able to restart the network, although a reboot is recommended to see if everything works properly.

```
$ sudo service networking restart
$ sudo reboot
```

Avertissement : Make sure you have an alternative way like IPMI or ILO to reach the machine in case you made a configuration error and the network stops functioning !

Setup zone using VXLAN

In almost all parts of zone setup, you can just follow the advanced zone setup instruction in “PRODUCT Installation Guide” to use this plugin. It is not required to add a network element nor to reconfigure the network offering. The only thing you have to do is configure the physical network to use VXLAN as the isolation method for Guest Network.

Configure the physical network

CloudStack needs to have one physical network for Guest Traffic with the isolation method set to “VXLAN”.

Guest Network traffic label should be the name of the physical interface or the name of the bridge interface and the bridge interface and they should have an IPv4 address. See ? for details.

Configure the guest traffic

Specify a range of VNIs you would like to use for carrying guest network traffic.

Avertissement : VNI must be unique per zone and no duplicate VNIs can exist in the zone. Exercise care when designing your VNI allocation policy.

The OVS Plugin

Introduction to the OVS Plugin

The OVS plugin is the native SDN implementations in CloudStack, using GRE isolation method. The plugin can be used by CloudStack to implement isolated guest networks and to provide additional services like NAT, port forwarding and load balancing.

 Add zone


- 1 Zone Type
- 2 Setup Zone
- 3 Setup Network
- 4 Add Resources
- 5 Launch


[PHYSICAL NETWORK >](#) [PUBLIC TRAFFIC >](#) [POD >](#) [GUEST TRAFFIC >](#)


When adding an advanced zone, you need to set up one or more physical networks. Each network corresponds to a NIC on the hypervisor. Each physical network can carry one or more types of traffic, with certain restrictions on how they may be combined.

Drag and drop one or more traffic types onto each physical network.

Traffic Types



 Guest



 Storage




Physical network name

Isolation method VLAN ▾



 Management
Edit


 Public
Edit


 Storage
Edit

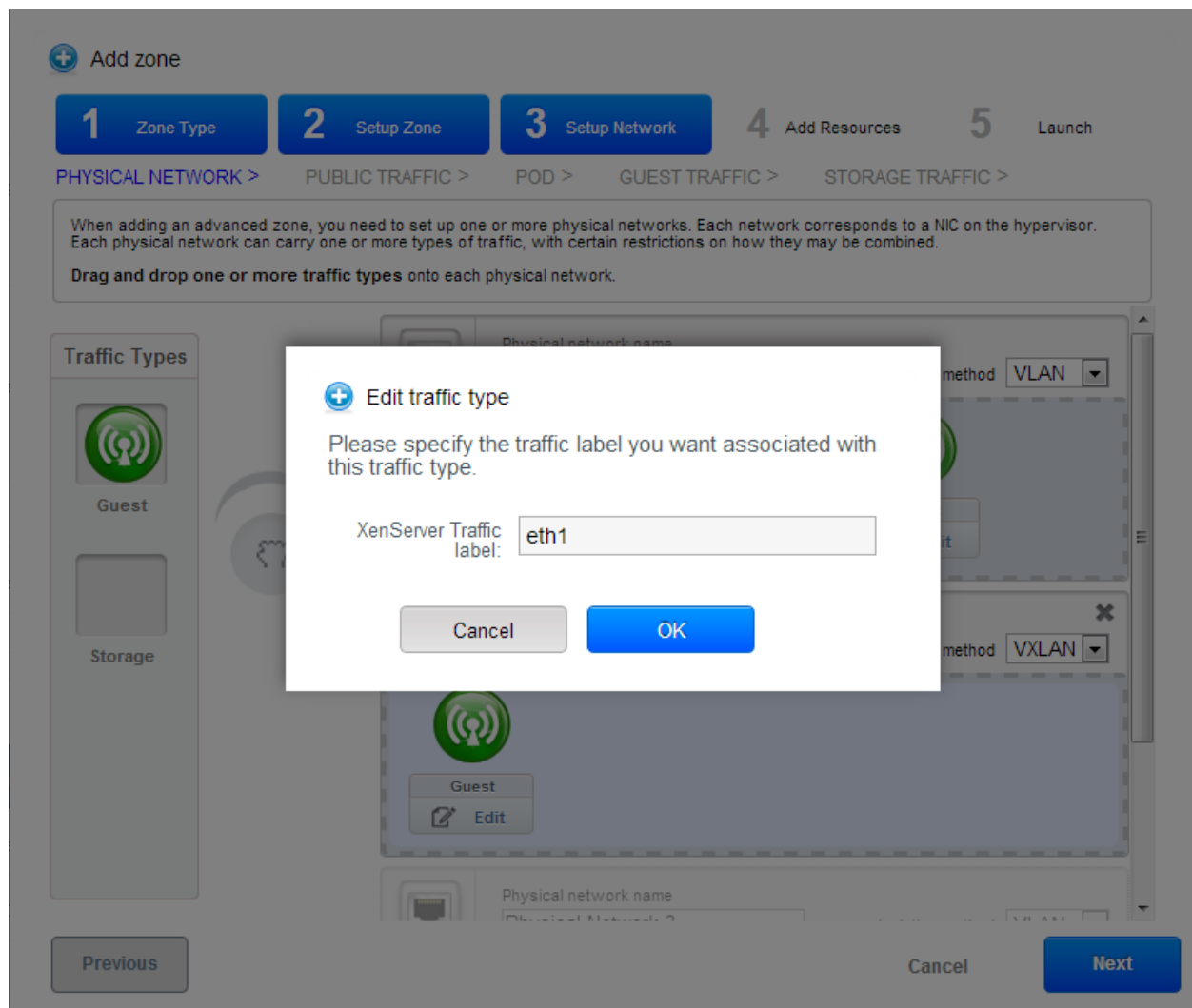
Physical network name


Isolation method VXLAN ▾


 Guest
Edit

Physical network name

Previous
Cancel
Next



 Add zone

1 Zone Type

2 Setup Zone

3 Setup Network

4 Add Resources

5 Launch

[PUBLIC TRAFFIC >](#) [POD >](#) [GUEST TRAFFIC >](#) [STORAGE TRAFFIC >](#)

Guest network traffic is communication between end-user virtual machines. Specify a range of VLAN IDs to carry guest traffic for each physical network.

Physical Network 2

VLAN/VNI Range:

Previous

Cancel

Next

Features of the OVS Plugin

The following table lists the CloudStack network services provided by the OVS Plugin.

Network Service	CloudStack version
Virtual Networking	>= 4.0
Static NAT	>= 4.3
Port Forwarding	>= 4.3
Load Balancing	>= 4.3

Table : Supported Services

Note : The Virtual Networking service was originally called ‘Connectivity’ in CloudStack 4.0

The following hypervisors are supported by the OVS Plugin.

Hypervisor	CloudStack version
XenServer	>= 4.0
KVM	>= 4.3

Table : Supported Hypervisors

Configuring the OVS Plugin

Prerequisites

Before enabling the OVS plugin the hypervisor needs to be install OpenvSwitch. Default, XenServer has already installed OpenvSwitch. However, you must install OpenvSwitch manually on KVM. CentOS 6.4 and OpenvSwitch 1.10 are recommended.

KVM hypervisor :

- CentOS 6.4 is recommended.
- To make sure that the native bridge module will not interfere with openvSwitch the bridge module should be added to the blacklist. See the modprobe documentation for your distribution on where to find the blacklist. Make sure the module is not loaded either by rebooting or executing `rmmod bridge` before executing next steps.

Zone Configuration

CloudStack needs to have at least one physical network with the isolation method set to “GRE”. This network should be enabled for the Guest traffic type.

Note : With KVM, the traffic type should be configured with the traffic label that matches the name of the Integration Bridge on the hypervisor. For example, you should set the traffic label as following : - Management & Storage traffic : `cloudbr0` - Guest & Public traffic : `cloudbr1` See KVM networking configuration guide for more detail.

Agent Configuration



Note : Only for KVM hypervisor

- Configure network interfaces :

Home > Infrastructure > Zones > z >

Refresh

Details Compute and Storage **Physical Network** Resources System VMs

Name	State	Isolation method	Actions
Physical Network 1	 Enabled	GRE	

```
/etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=none
IPV6INIT=no
NM_CONTROLLED=no
ONBOOT=yes
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=cloudbr0

/etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
BOOTPROTO=none
IPV6INIT=no
NM_CONTROLLED=no
ONBOOT=yes
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=cloudbr1

/etc/sysconfig/network-scripts/ifcfg-cloudbr0
DEVICE=cloudbr0
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=172.16.10.10
GATEWAY=172.16.10.1
NETMASK=255.255.255.0
HOTPLUG=no

/etc/sysconfig/network-scripts/ifcfg-cloudbr1
DEVICE=cloudbr1
ONBOOT=yes
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=none
HOTPLUG=no

/etc/sysconfig/network
NETWORKING=yes
HOSTNAME=testkvm1
```

```
GATEWAY=172.10.10.1
```














— Edit /etc/cloudstack/agent/agent.properties

```
network.bridge.type=openvswitch
libvirt.vif.driver=com.cloud.hypervisor.kvm.resource.OvsVifDriver
```

Enabling the service provider

The OVS provider is disabled by default. Navigate to the “Network Service Providers” configuration of the physical network with the GRE isolation type. Navigate to the OVS provider and press the “Enable Provider” button.

[Home](#) > [Infrastructure](#) > [Zones](#) > [z](#) > [Physical Network 1](#) > [Network Service Providers](#) >

Name	State
NetScaler	 Disabled
Virtual Router	 Enabled
Nicira Nvp	 Disabled
BigSwitch Vns	 Disabled
Baremetal DHCP	 Disabled
Baremetal PXE	 Disabled
Ovs	 Enabled
Cisco VNMC	 Absent
MidoNet	 Disabled
Internal LB VM	 Enabled
VPC Virtual Router	 Enabled
F5	 Disabled
SRX	 Disabled

Network Offerings

Using the OVS plugin requires a network offering with Virtual Networking enabled and configured to use the OVS element. Typical use cases combine services from the Virtual Router appliance and the OVS plugin.

Service	Provider
VPN	VirtualRouter
DHCP	VirtualRouter
DNS	VirtualRouter
Firewall	VirtualRouter
Load Balancer	OVS
User Data	VirtualRouter
Source NAT	VirtualRouter
Static NAT	OVS
Port Forwarding	OVS
Virtual Networking	OVS

Table : Isolated network offering with regular services from the Virtual Router.

Add network offering

* Name:

* Description:

Network Rate (Mb/s):

Guest Type:

Persistent : ☐

Specify VLAN: ☐

VPC: ☐

Supported Services:

Static NAT Provider:	<input type="text" value="Ovs"/>
Port Forwarding:	<input checked="" type="checkbox"/>
Port Forwarding Provider:	<input type="text" value="Ovs"/>
Security Groups:	<input type="checkbox"/>
NetworkACL:	<input type="checkbox"/>
Virtual Networking:	<input checked="" type="checkbox"/>
Virtual Networking Provider:	<input type="text" value="Ovs"/>

System Offering for Router:

Redundant router capability: ☐

Supported Source NAT type:

Note : The tag in the network offering should be set to the name of the physical network with the OVS provider.

Isolated network with network services. The virtual router is still required to provide network services like dns and dhcp.

Service	Provider
DHCP	VirtualRouter
DNS	VirtualRouter
User Data	VirtualRouter
Source NAT	VirtualRouter
Static NAT	OVS
Post Forwarding	OVS
Load Balancing	OVS
Virtual Networking	OVS

Table : Isolated network offering with network services

Using the OVS plugin with VPC

OVS plugin does not work with VPC at that time

Revision History

0-0 Mon Dec 2 2013 Nguyen Anh Tu tuna@apache.org Documentation created for 4.3.0 version of the OVS Plugin

IPv6 Support in CloudStack

CloudStack supports Internet Protocol version 6 (IPv6), the recent version of the Internet Protocol (IP) that defines routing the network traffic. IPv6 uses a 128-bit address that exponentially expands the current address space that is available to the users. IPv6 addresses consist of eight groups of four hexadecimal digits separated by colons, for example, 5001:0dt8:83a3:1012:1000:8s2e:0870:7454. CloudStack supports IPv6 for public IPs in shared networks. With IPv6 support, VMs in shared networks can obtain both IPv4 and IPv6 addresses from the DHCP server. You can deploy VMs either in a IPv6 or IPv4 network, or in a dual network environment. If IPv6 network is used, the VM generates a link-local IPv6 address by itself, and receives a stateful IPv6 address from the DHCPv6 server.

IPv6 is supported only on KVM and XenServer hypervisors. The IPv6 support is only an experimental feature.

Here's the sequence of events when IPv6 is used :

1. The administrator creates an IPv6 shared network in an advanced zone.
2. The user deploys a VM in an IPv6 shared network.
3. The user VM generates an IPv6 link local address by itself, and gets an IPv6 global or site local address through DHCPv6.

Prerequisites and Guidelines

Consider the following :

- CIDR size must be 64 for IPv6 networks.
- The DHCP client of the guest VMs should support generating DUID based on Link-layer Address (DUID-LL). DUID-LL derives from the MAC address of guest VMs, and therefore the user VM can be identified by using DUID. See [Dynamic Host Configuration Protocol for IPv6](#) for more information.

- The gateway of the guest network generates Router Advisement and Response messages to Router Solicitation. The M (Managed Address Configuration) flag of Router Advisement should enable stateful IP address configuration. Set the M flag to where the end nodes receive their IPv6 addresses from the DHCPv6 server as opposed to the router or switch.

Note : The M flag is the 1-bit Managed Address Configuration flag for Router

Advisement. When set, Dynamic Host Configuration Protocol (DHCPv6) is available for address configuration in addition to any IPs set by using stateless address auto-configuration.

- Use the System VM template exclusively designed to support IPv6. Download the System VM template from <http://cloudstack.appt-get.eu/systemvm/>.
- The concept of Default Network applies to IPv6 networks. However, unlike IPv4 CloudStack does not control the routing information of IPv6 in shared network ; the choice of Default Network will not affect the routing in the user VM.
- In a multiple shared network, the default route is set by the rack router, rather than the DHCP server, which is out of CloudStack control. Therefore, in order for the user VM to get only the default route from the default NIC, modify the configuration of the user VM, and set non-default NIC's `accept_ra` to 0 explicitly. The `accept_ra` parameter accepts Router Advertisements and auto-configure `/proc/sys/net/ipv6/conf/interface` with received data.

Limitations of IPv6 in CloudStack

The following are not yet supported :

1. Security groups
2. Userdata and metadata
3. Passwords

Guest VM Configuration for DHCPv6

For the guest VMs to get IPv6 address, run `dhclient` command manually on each of the VMs. Use DUID-LL to set up `dhclient`.

Note : The IPv6 address is lost when a VM is stopped and started. Therefore,

use the same procedure to get an IPv6 address when a VM is stopped and started.

1. Set up `dhclient` by using DUID-LL.
Perform the following for DHCP Client 4.2 and above :
 - (a) Run the following command on the selected VM to get the `dhcpv6` offer from VR :

```
dhclient -6 -D LL <dev>
```

Perform the following for DHCP Client 4.1 :

- (a) Open the following to the `dhclient` configuration file :

```
vi /etc/dhcp/dhclient.conf
```

- (b) Add the following to the `dhclient` configuration file :

```
send dhcp6.client-id = concat(00:03:00, hardware);
```

2. Get IPv6 address from DHCP server as part of the system or network restart.

Based on the operating systems, perform the following :

On CentOS 6.2 :

- (a) Open the Ethernet interface configuration file :

```
vi /etc/sysconfig/network-scripts/ifcfg-eth0
```

The ifcfg-eth0 file controls the first NIC in a system.

- (b) Make the necessary configuration changes, as given below :

```
DEVICE=eth0
HWADDR=06:A0:F0:00:00:38
NM_CONTROLLED=no
ONBOOT=yes
BOOTPROTO=dhcp6
TYPE=Ethernet
USERCTL=no
PEERDNS=yes
IPV6INIT=yes
DHCPV6C=yes
```

- (c) Open the following :

```
vi /etc/sysconfig/network
```

- (d) Make the necessary configuration changes, as given below :

```
NETWORKING=yes
HOSTNAME=centos62mgmt.lab.vmops.com
NETWORKING_IPV6=yes
IPV6_AUTOCONF=no
```

On Ubuntu 12.10

- (a) Open the following :

```
etc/network/interfaces:
```

- (b) Make the necessary configuration changes, as given below :

```
iface eth0 inet6 dhcp
autoconf 0
accept_ra 1
```

Configuring AutoScale without using NetScaler

Avertissement : This feature is currently only available on the master branch and will be released in the 4.4 release.

What is AutoScaling ?

AutoScaling allows you to scale your back-end services or application VMs up or down seamlessly and automatically according to the conditions you define. With AutoScaling enabled, you can ensure that the number of VMs you are using seamlessly scale up when demand increases, and automatically decreases when demand subsides. Thus it helps you save compute costs by terminating underused VMs automatically and launching new VMs when you need them, without the need for manual intervention.

Hypervisor support

At that time, AutoScaling without NetScaler only supports for Xenserver. We are working to support KVM also.

Prerequisites

Before you configure an AutoScale rule, consider the following :

- Ensure that the necessary template is prepared before configuring AutoScale. Firstly you must install the PV-driver, which helps Xenserver collect performance parameters (CPU and memory) into VMs. Beside, When a VM is deployed by using a template and when it comes up, the application should be up and running.

Configuration

Specify the following :

AutoScale Configuration Wizard

Template:

Compute offering:

* Min Instances: * Max Instances:

Scale Up Policy

* Duration(in sec):

Counter	Operator	Threshold	Add
<input type="text" value="Linux User CPU - percentage"/>	<input type="text" value="greater-than"/>	<input type="text" value="1000"/>	<input type="button" value="Add"/>
<input type="text" value="Response Time - microseconds"/>	<input type="text" value="greater-than"/>	<input type="text" value="1000"/>	<input type="button" value="X"/>

Scale Down Policy

* Duration(in sec):

Counter	Operator	Threshold	Add
<input type="text" value="Linux User CPU - percentage"/>	<input type="text" value="greater-than"/>	<input type="text" value="1000"/>	<input type="button" value="Add"/>

- Template : A template consists of a base OS image and application. A template is used to provision the new instance of an application on a scaleup action. When a VM is deployed from a template, the VM can start taking the traffic from the load balancer without any admin intervention. For example, if the VM is deployed for a Web service, it should have the Web server running, the database connected, and so on.

- **Compute offering** : A predefined set of virtual hardware attributes, including CPU speed, number of CPUs, and RAM size, that the user can select when creating a new virtual machine instance. Choose one of the compute offerings to be used while provisioning a VM instance as part of scaleup action.
- **Min Instance** : The minimum number of active VM instances that is assigned to a load balancing rule. The active VM instances are the application instances that are up and serving the traffic, and are being load balanced. This parameter ensures that a load balancing rule has at least the configured number of active VM instances are available to serve the traffic.
- **Max Instance** : Maximum number of active VM instances that should be assigned to a load balancing rule. This parameter defines the upper limit of active VM instances that can be assigned to a load balancing rule.

Specifying a large value for the maximum instance parameter might result in provisioning large number of VM instances, which in turn leads to a single load balancing rule exhausting the VM instances limit specified at the account or domain level.

Specify the following scale-up and scale-down policies :

- **Duration** : The duration, in seconds, for which the conditions you specify must be true to trigger a scaleup action. The conditions defined should hold true for the entire duration you specify for an AutoScale action to be invoked.
- **Counter** : The performance counters expose the state of the monitored instances. We added two new counter to work with that feature :
 - Linux User CPU [native] - percentage
 - Linux User RAM [native] - percentage

Remember to choose one of them. If you choose anything else, the autoscaling will not work.

- **Operator** : The following five relational operators are supported in AutoScale feature : Greater than, Less than, Less than or equal to, Greater than or equal to, and Equal to.
- **Threshold** : Threshold value to be used for the counter. Once the counter defined above breaches the threshold value, the AutoScale feature initiates a scaleup or scaledown action.
- **Add** : Click Add to add the condition.

Additionally, if you want to configure the advanced settings, click Show advanced settings, and specify the following :

- **Polling interval** : Frequency in which the conditions, combination of counter, operator and threshold, are to be evaluated before taking a scale up or down action. The default polling interval is 30 seconds.
- **Quiet Time** : This is the cool down period after an AutoScale action is initiated. The time includes the time taken to complete provisioning a VM instance from its template and the time taken by an application to be ready to serve traffic. This quiet time allows the fleet to come up to a stable state before any action can take place. The default is 300 seconds.
- **Destroy VM Grace Period** : The duration in seconds, after a scaledown action is initiated, to wait before the VM is destroyed as part of scaledown action. This is to ensure graceful close of any pending sessions or transactions being served by the VM marked for destroy. The default is 120 seconds.
- **Apply** : Click Apply to create the AutoScale configuration.

Disabling and Enabling an AutoScale Configuration

If you want to perform any maintenance operation on the AutoScale VM instances, disable the AutoScale configuration. When the AutoScale configuration is disabled, no scaleup or scaledown action is performed. You can use this downtime for the maintenance activities. To disable the AutoScale configuration, click the Disable AutoScale button.

The button toggles between enable and disable, depending on whether AutoScale is currently enabled or not. After the maintenance operations are done, you can enable the AutoScale configuration back. To enable, open the AutoScale configuration page again, then click the Enable AutoScale button.

Updating an AutoScale Configuration

You can update the various parameters and add or delete the conditions in a scaleup or scaledown rule. Before you update an AutoScale configuration, ensure that you disable the AutoScale load balancer rule by clicking the Disable AutoScale button. After you modify the required AutoScale parameters, click Apply. To apply the new AutoScale policies, open the AutoScale configuration page again, then click the Enable AutoScale button.

Runtime Considerations

An administrator should not assign a VM to a load balancing rule which is configured for AutoScale.

Making API calls outside the context of AutoScale, such as `destroyVM`, on an autoscaled VM leaves the load balancing configuration in an inconsistent state. Though VM is destroyed from the load balancer rule, it continues be showed as a service assigned to a rule inside the context of AutoScale.

CloudStack Installation from Source for Developers

This book is aimed at CloudStack developers who need to build the code. These instructions are valid on a Ubuntu 12.04 and CentOS 6.4 systems and were tested with the 4.2 release of Apache CloudStack, please adapt them if you are on a different operating system or using a newer/older version of CloudStack. This book is composed of the following sections :

1. Installation of the prerequisites
2. Compiling and installation from source
3. Using the CloudStack simulator
4. Installation with DevCloud the CloudStack sandbox
5. Building your own packages
6. The CloudStack API
7. Testing the AWS API interface

Prerequisites

In this section we'll look at installing the dependencies you'll need for Apache CloudStack development.

On Ubuntu 12.04

First update and upgrade your system :

```
apt-get update  
apt-get upgrade
```

NTP might already be installed, check it with `service ntp status`. If it's not then install NTP to synchronize the clocks :

```
apt-get install openntpd
```

Install `openjdk`. As we're using Linux, OpenJDK is our first choice.

```
apt-get install openjdk-6-jdk
```

Install `tomcat6`, note that the new version of tomcat on [Ubuntu](#) is the 6.0.35 version.

```
apt-get install tomcat6
```

Next, we'll install MySQL if it's not already present on the system.

```
apt-get install mysql-server
```

Remember to set the correct `mysql` password in the CloudStack properties file. Mysql should be running but you can check it's status with :

```
service mysql status
```

Developers wanting to build CloudStack from source will want to install the following additional packages. If you don't want to build from source just jump to the next section.

Install `git` to later clone the CloudStack source code :

```
apt-get install git
```

Install `Maven` to later build CloudStack

```
apt-get install maven
```

This should have installed `Maven 3.0`, check the version number with `mvn --version`

A little bit of Python can be used (e.g simulator), install the Python package management tools :

```
apt-get install python-pip python-setuptools
```

Finally install `mkisofs` with :

```
apt-get install genisoimage
```

On CentOS 6.4

First update and upgrade your system :

```
yum -y update  
yum -y upgrade
```

If not already installed, install NTP for clock synchronization

```
yum -y install ntp
```

Install `openjdk`. As we're using Linux, OpenJDK is our first choice.

```
yum -y install java-1.6.0-openjdk
```


Install tomcat6, note that the version of tomcat6 in the default CentOS 6.4 repo is 6.0.24, so we will grab the 6.0.35 version. The 6.0.24 version will be installed anyway as a dependency to cloudstack.

```
wget https://archive.apache.org/dist/tomcat/tomcat-6/v6.0.35/bin/apache-tomcat-6.0.35.
→tar.gz
tar xzvf apache-tomcat-6.0.35.tar.gz -C /usr/local
```

Setup tomcat6 system wide by creating a file /etc/profile.d/tomcat.sh with the following content :

```
export CATALINA_BASE=/usr/local/apache-tomcat-6.0.35
export CATALINA_HOME=/usr/local/apache-tomcat-6.0.35
```

Next, we'll install MySQL if it's not already present on the system.

```
yum -y install mysql mysql-server
```

Remember to set the correct mysql password in the CloudStack properties file. Mysql should be running but you can check it's status with :

```
service mysqld status
```

Install git to later clone the CloudStack source code :

```
yum -y install git
```

Install Maven to later build CloudStack. Grab the 3.0.5 release from the Maven [website](#)

```
wget http://mirror.cc.columbia.edu/pub/software/apache/maven/maven-3/3.0.5/binaries/
→apache-maven-3.0.5-bin.tar.gz
tar xzf apache-maven-3.0.5-bin.tar.gz -C /usr/local
cd /usr/local
ln -s apache-maven-3.0.5 maven
```

Setup Maven system wide by creating a /etc/profile.d/maven.sh file with the following content :

```
export M2_HOME=/usr/local/maven
export PATH=${M2_HOME}/bin:${PATH}
```

Log out and log in again and you will have maven in your PATH :

```
mvn --version
```

This should have installed Maven 3.0, check the version number with mvn --version

A little bit of Python can be used (e.g simulator), install the Python package management tools :

```
yum -y install python-setuptools
```

To install python-pip you might want to setup the Extra Packages for Enterprise Linux (EPEL) repo

```
cd /tmp
wget http://mirror-fpt-telecom.fpt.net/fedora/epel/6/i386/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm
```

Then update you repository cache yum update and install pip yum -y install python-pip

Finally install mkisofs with :

```
yum -y install genisoimage
```

Installing from Source

CloudStack uses git for source version control, if you know little about [git](#) is a good start. Once you have git setup on your machine, pull the source with :

```
git clone https://git-wip-us.apache.org/repos/asf/cloudstack.git
```

To build the latest stable release :

```
git checkout 4.2
```

To compile Apache CloudStack, go to the cloudstack source folder and run :

```
mvn -Pdeveloper,systemvm clean install
```

If you want to skip the tests add `-DskipTests` to the command above. Do NOT use `-Dmaven.test.skip=true` because that will break the build.

You will have made sure to set the proper db password in `utils/conf/db.properties`

Deploy the database next :

```
mvn -P developer -pl developer -Ddeploydb
```

Run Apache CloudStack with jetty for testing. Note that tomcat maybe be running on port 8080, stop it before you use jetty

```
mvn -pl :cloud-client-ui jetty:run
```

Log Into Apache CloudStack :

Open your Web browser and use this URL to connect to CloudStack :

```
http://localhost:8080/client/
```

Replace `localhost` with the IP of your management server if need be.

Note : If you have iptables enabled, you may have to open the ports used by CloudStack. Specifically, ports 8080, 8250, and 9090.

You can now start configuring a Zone, playing with the API. Of course we did not setup any infrastructure, there is no storage, no hypervisors...etc. However you can run tests using the simulator. The following section shows you how to use the simulator so that you don't have to setup a physical infrastructure.

Using the Simulator

CloudStack comes with a simulator based on Python bindings called *Marvin*. Marvin is available in the CloudStack source code or on Pypi. With Marvin you can simulate your data center infrastructure by providing CloudStack with a configuration file that defines the number of zones/pods/clusters/hosts, types of storage etc. You can then develop and test the CloudStack management server *as if* it was managing your production infrastructure.

Do a clean build :

```
mvn -Pdeveloper -Dsimulator -DskipTests clean install
```

Deploy the database :

```
mvn -Pdeveloper -pl developer -Ddeploydb
mvn -Pdeveloper -pl developer -Ddeploydb-simulator
```

Install marvin. Note that you will need to have installed pip properly in the prerequisites step.

```
pip install tools/marvin/dist/Marvin-0.1.0.tar.gz
```

Stop jetty (from any previous runs)

```
mvn -pl :cloud-client-ui jetty:stop
```

Start jetty

```
mvn -pl client jetty:run
```

Setup a basic zone with Marvin. In a separate shell ://

```
mvn -Pdeveloper,marvin.setup -Dmarvin.config=setup/dev/basic.cfg -pl :cloud-marvin_
↪integration-test
```

At this stage log in the CloudStack management server at <http://localhost:8080/client> with the credentials admin/password, you should see a fully configured basic zone infrastructure. To simulate an advanced zone replace `basic.cfg` with `advanced.cfg`.

You can now run integration tests, use the API etc...

Using DevCloud

The Installing from source section will only get you to the point of running the management server, it does not get you any hypervisors. The simulator section gets you a simulated datacenter for testing. With DevCloud you can run at least one hypervisor and add it to your management server the way you would a real physical machine.

DevCloud is the CloudStack sandbox, the standard version is a VirtualBox based image. There is also a KVM based image for it. Here we only show steps with the VirtualBox image. For KVM see the [wiki](#).

**** DevCloud Pre-requisites**

1. Install [VirtualBox](#) on your machine
2. Run VirtualBox and under >Preferences create a *host-only interface* on which you disable the DHCP server
3. Download the DevCloud [image](#)
4. In VirtualBox, under File > Import Appliance import the DevCloud image.
5. Verify the settings under > Settings and check the enable PAE option in the processor menu
6. Once the VM has booted try to ssh to it with credentials : root/password
ssh root@192.168.56.10

Adding DevCloud as an Hypervisor

Picking up from a clean build :

```
mvn -Pdeveloper,systemvm clean install
mvn -P developer -pl developer,tools/devcloud -Ddeploydb
```

At this stage install marvin similarly than with the simulator :

```
pip install tools/marvin/dist/Marvin-0.1.0.tar.gz
```

Start the management server

```
mvn -pl client jetty:run
```

Then you are going to configure CloudStack to use the running DevCloud instance :

```
cd tools/devcloud
python ../marvin/marvin/deployDataCenter.py -i devcloud.cfg
```

If you are curious, check the `devcloud.cfg` file and see how the data center is defined : 1 Zone, 1 Pod, 1 Cluster, 1 Host, 1 primary Storage, 1 Secondary Storage, all provided by Devcloud.

You can now log in the management server at `http://localhost:8080/client` and start experimenting with the UI or the API.

Do note that the management server is running in your local machine and that DevCloud is used only as a n Hypervisor. You could potentially run the management server within DevCloud as well, or memory granted, run multiple DevClouds.

Building Packages

Working from source is necessary when developing CloudStack. As mentioned earlier this is not primarily intended for users. However some may want to modify the code for their own use and specific infrastructure. The may also need to build their own packages for security reasons and due to network connectivity constraints. This section shows you the gist of how to build packages. We assume that the reader will know how to create a repository to serve this packages. The complete documentation is available on the [website](#)

To build debian packages you will need couple extra packages that we did not need to install for source compilation :

```
apt-get install python-mysqldb
apt-get install debhelper
```

Then build the packages with :

```
dpkg-buildpackage -uc -us
```

One directory up from the CloudStack root dir you will find :

```
cloudstack_4.2.0_amd64.changes
cloudstack_4.2.0.dsc
cloudstack_4.2.0.tar.gz
cloudstack-agent_4.2.0_all.deb
cloudstack-awsapi_4.2.0_all.deb
cloudstack-cli_4.2.0_all.deb
cloudstack-common_4.2.0_all.deb
cloudstack-docs_4.2.0_all.deb
cloudstack-management_4.2.0_all.deb
cloudstack-usage_4.2.0_all.deb
```

Of course the community provides a repository for these packages and you can use it instead of building your own packages and putting them in your own repo. Instructions on how to use this community repository are available in the installation book.

The CloudStack API

The CloudStack API is a query based API using http that return results in XML or JSON. It is used to implement the default web UI. This API is not a standard like [OGF OCCI](#) or [DMTF CIMI](#) but is easy to learn. Mapping exists between the AWS API and the CloudStack API as will be seen in the next section. Recently a Google Compute Engine interface was also developed that maps the GCE REST API to the CloudStack API described here. The [API docs](#) are a good start to learn the extent of the API. Multiple clients exist on [github](#) to use this API, you should be able to find one in your favorite language. The reference documentation for the API and changes that might occur from version to version is available [on-line](#). This short section is aimed at providing a quick summary to give you a base understanding of how to use this API. As a quick start, a good way to explore the API is to navigate the dashboard with a firebug console (or similar developer console) to study the queries.

In a succinct statement, the CloudStack query API can be used via http GET requests made against your cloud endpoint (e.g <http://localhost:8080/client/api>). The API name is passed using the `command` key and the various parameters for this API call are passed as key value pairs. The request is signed using the access key and secret key of the user making the call. Some calls are synchronous while some are asynchronous, this is documented in the [API docs](#). Asynchronous calls return a `jobid`, the status and result of a job can be queried with the `queryAsyncJobResult` call. Let's get started and give an example of calling the `listUsers` API in Python.

First you will need to generate keys to make requests. Going through the dashboard, go under `Accounts` select the appropriate account then click on `Show Users` select the intended users and generate keys using the `Generate Keys` icon. You will see an `API Key` and `Secret Key` field being generated. The keys will be of the form :

```
API Key : XzAz0uC0t888gOzPs3HchY72qwDc7pUPI08LxC-VkIH04C3fvbEBY_Ccj8fo3mBapN5qRDg_0_
↳EbGdbxi8oylA
Secret Key: zmBOXAXPlfb-LIygOxUVblAbz7E47eukDS_0JYUxP3JAmknOYo56T0R-
↳AcM7rK7SMYo1lY6XW22gyuXzOdiyBQ
```

Open a Python shell and import the basic modules necessary to make the request. Do note that this request could be made many different ways, this is just a low level example. The `urllib*` modules are used to make the http request and do url encoding. The `hashlib` module gives us the `sha1` hash function. It used to generate the `hmac` (Keyed Hashing for Message Authentication) using the `secretkey`. The result is encoded using the `base64` module.

```
$python
Python 2.7.3 (default, Nov 17 2012, 19:54:34)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apples/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> import urllib
>>> import hashlib
>>> import hmac
>>> import base64
```

Define the endpoint of the Cloud, the command that you want to execute, the type of the response (i.e XML or JSON) and the keys of the user. Note that we do not put the `secretkey` in our request dictionary because it is only used to compute the `hmac`.

```
>>> baseurl='http://localhost:8080/client/api?'
>>> request={}
>>> request['command']='listUsers'
>>> request['response']='json'
```

```
>>> request['apikey']='plgWJfZK4gyS3mOMTVmjUVg-X-jlWlnfaUJ9GAbBbf9EdM-
↳kAYMmAiLqzzq1ElZLYq_u38zCm0bewzGUDP66mg'
>>> secretkey='VDaACYb0LV9eNjTetIOElcVQkvJck_J_Q1jX_
↳FcHRj87ZKiy0z0ty0ZsYBkoXkY9b7eq1EhwJaw7FF3akA3KBQ'
```

Build the base request string, the combination of all the key/pairs of the request, url encoded and joined with ampersand.

```
>>> request_str='&'.join(['=' .join([k,urllib.quote_plus(request[k])]) for k in_
↳request.keys()])
>>> request_str
'apikey=plgWJfZK4gyS3mOMTVmjUVg-X-jlWlnfaUJ9GAbBbf9EdM-kAYMmAiLqzzq1ElZLYq_
↳u38zCm0bewzGUDP66mg&command=listUsers&response=json'
```

Compute the signature with hmac, do a 64 bit encoding and a url encoding, the string used for the signature is similar to the base request string shown above but the keys/values are lower cased and joined in a sorted order

```
>>> sig_str='&'.join(['=' .join([k.lower(),urllib.quote_plus(request[k].lower())
↳replace('+','%20')]) for k in sorted(request.iterkeys())])
>>> sig_str
'apikey=plgwjfk4gys3momtvmjuvg-x-jlwlnfauj9gabbbf9edm-kaymmailqzzq1elzlyq_
↳u38zcm0bewzgudp66mg&command=listusers&response=json'
>>> sig=hmac.new(secretkey,sig_str,hashlib.sha1).digest()
>>> sig
'M:]\x0e\xaf\xfb\x8f\x2y\xflp\x91\x1e\x89\x8a\xa1\x05\xc4A\xdb'
>>> sig=base64.encodestring(hmac.new(secretkey,sig_str,hashlib.sha1).digest())
>>> sig
'TTpdDq/7j/J58XCRHomKoQXEQds=\n'
>>> sig=base64.encodestring(hmac.new(secretkey,sig_str,hashlib.sha1).digest()).strip()
>>> sig
'TTpdDq/7j/J58XCRHomKoQXEQds='
>>> sig=urllib.quote_plus(base64.encodestring(hmac.new(secretkey,sig_str,hashlib.
↳sha1).digest()).strip())
```

Finally, build the entire string by joining the baseurl, the request str and the signature. Then do an http GET :

```
>>> req=baseurl+request_str+'&signature='+sig
>>> req
'http://localhost:8080/client/api?apikey=plgWJfZK4gyS3mOMTVmjUVg-X-
↳jlWlnfaUJ9GAbBbf9EdM-kAYMmAiLqzzq1ElZLYq_u38zCm0bewzGUDP66mg&command=listUsers&
↳response=json&signature=TTpdDq%2F7j%2FJ58XCRHomKoQXEQds%3D'
>>> res=urllib2.urlopen(req)
>>> res.read()
'{"listusersresponse" : { "count":1 , "user" : [ {"id":"7ed6d5da-93b2-4545-a502-
↳23d20b48ef2a", "username":"admin", "firstname":"admin",
↳"lastname":"cloud", "created":"2012-
↳07-05T12:18:27-0700", "state":"enabled", "account":"admin",
↳"accounttype":1, "domainid":
↳"8a111e58-e155-4482-93ce-84efff3c7c77", "domain":"ROOT",
↳"apikey":"plgWJfZK4gyS3mOMTVmjUVg-
↳X-jlWlnfaUJ9GAbBbf9EdM-kAYMmAiLqzzq1ElZLYq_u38zCm0bewzGUDP66mg",
↳"secretkey":
↳"VDaACYb0LV9eNjTetIOElcVQkvJck_J_Q1jX_
↳FcHRj87ZKiy0z0ty0ZsYBkoXkY9b7eq1EhwJaw7FF3akA3KBQ",
↳"accountid":"7548ac03-af1d-4c1c-
↳9064-2f3e2c0eda0d"} ] } }
```

All the clients that you will find on github will implement this signature technique, you should not have to do it by

hand. Now that you have explored the API through the UI and that you understand how to make low level calls, pick your favorite client of use [CloudMonkey](#). CloudMonkey is a sub-project of Apache CloudStack and gives operators/developers the ability to use any of the API methods. It has nice auto-completion and help feature as well as an API discovery mechanism since 4.2.

Testing the AWS API interface

While the native CloudStack API is not a standard, CloudStack provides a AWS EC2 compatible interface. It has the great advantage that existing tools written with EC2 libraries can be re-used against a CloudStack based cloud. In the installation books we described how to run this interface from installing packages. In this section we show you how to compile the interface with maven and test it with Python boto module.

Starting from a running management server (with DevCloud for instance), start the AWS API interface in a separate shell with :

```
mvn -Pawsapi -pl :cloud-awsapi jetty:run
```

Log into the CloudStack UI <http://localhost:8080/client>, go to *Service Offerings* and edit one of the compute offerings to have the name `m1.small` or any of the other AWS EC2 instance types.

With access and secret keys generated for a user you should now be able to use Python [Boto](#) module :

```
import boto
import boto.ec2

accesskey="2IUSA5xylbsPSnBQFoWXXKg3RvjHgsufcKhC1SeiCbeEc0obKwUlwJamB_
↳gFmMJkFHYHTIafpUx0pHcfLvt-dzw"
secretkey=
↳"0xV5Dhhk5ufNowey7OVHgWxCBVS4deTl9qL0EqMthfPBuy3ScHPo2fifDxwlaXeL5cyH10hnLOKjyKphcXGeDA
↳"

region = boto.ec2.regioninfo.RegionInfo(name="ROOT", endpoint="localhost")
conn = boto.connect_ec2(aws_access_key_id=accesskey, aws_secret_access_key=secretkey,
↳is_secure=False, region=region, port=7080, path="/awsapi", api_version="2012-08-15")

images=conn.get_all_images()
print images

res = images[0].run(instance_type='m1.small', security_groups=['default'])
```

Note the new `api_version` number in the connection object and also note that there was no user registration to make like in previous CloudStack releases.

Conclusions

CloudStack is a mostly Java application running with Tomcat and Mysql. It consists of a management server and depending on the hypervisors being used, an agent installed on the hypervisor farm. To complete a Cloud infrastructure however you will also need some Zone wide storage a.k.a Secondary Storage and some Cluster wide storage a.k.a Primary storage. The choice of hypervisor, storage solution and type of Zone (i.e Basic vs. Advanced) will dictate how complex your installation can be. As a quick start, you might want to consider KVM+NFS and a Basic Zone.

If you've run into any problems with this, please ask on the [cloudstack-dev](#) mailing list.

Programmer Guide

This guide shows how to develop CloudStack, use the API for operation and integration, access the usage data and use CloudStack specific tools to ease development, testing and integration.

The CloudStack API

Getting Started

To get started using the CloudStack API, you should have the following :

- URL of the CloudStack server you wish to integrate with.
- Both the API Key and Secret Key for an account. This should have been generated by the administrator of the cloud instance and given to you.
- Familiarity with HTTP GET/POST and query strings.
- Knowledge of either XML or JSON.
- Knowledge of a programming language that can generate HTTP requests ; for example, Java or PHP.

Roles

The CloudStack API supports three access roles :

1. Root Admin. Access to all features of the cloud, including both virtual and physical resource management.
2. Domain Admin. Access to only the virtual resources of the clouds that belong to the administrator's domain.
3. User. Access to only the features that allow management of the user's virtual instances, storage, and network.

API Reference Documentation

You can find all the API reference documentation at the below site :

<http://cloudstack.apache.org/docs/api/>

Making API Requests

All CloudStack API requests are submitted in the form of a HTTP GET/POST with an associated command and any parameters. A request is composed of the following whether in HTTP or HTTPS :

- CloudStack API URL : This is the web services API entry point(for example, <http://www.cloud.com:8080/client/api>)
- Command : The web services command you wish to execute, such as start a virtual machine or create a disk volume
- Parameters : Any additional required or optional parameters for the command

A sample API GET request looks like the following :

```
http://localhost:8080/client/api?command=deployVirtualMachine&serviceOfferingId=1&diskOfferingId=1&
templateId=2&zoneId=4&apiKey=miVr6X7u6bN_sdahOBpjNejPgEsT35eXq-jB8CG20YI3yaxXcgpyuaIRmFI_
EJTVwZ0nUkkJbPmY3y2bciKwFQ&signature=Lxx1DM40AjcXU%2FcaiK8RAP0O1hU%3D
```

Or in a more readable format :

```
1. http://localhost:8080/client/api
2. ?command=deployVirtualMachine
3. &serviceOfferingId=1
4. &diskOfferingId=1
```



```

5. &templateId=2
6. &zoneId=4
7. &apiKey=miVr6X7u6bN_sdahOBpjNejPgEsT35eXqjB8CG20YI3yaxXcgpyuaIRmFI_
  ↪EJTVwZ0nUkkJbPmY3y2bciKwFQ
8. &signature=Lxx1DM40AjcXU%2FcaiK8RAP001hU%3D

```

The first line is the CloudStack API URL. This is the Cloud instance you wish to interact with.

The second line refers to the command you wish to execute. In our example, we are attempting to deploy a fresh new virtual machine. It is preceded by a (?) to separate itself from the CloudStack API URL.

Lines 3-6 are the parameters for this given command. To see the command and its request parameters, please refer to the appropriate section in the CloudStack API documentation. Each parameter field-value pair (field=value) is preceded by an ampersand character (&).

Line 7 is the user API Key that uniquely identifies the account. See Signing API Requests on page 7.

Line 8 is the signature hash created to authenticate the user account executing the API command.

Signing API Requests

Whether you access the CloudStack API with HTTP or HTTPS, it must still be signed so that CloudStack can verify the caller has been authenticated and authorized to execute the command. Make sure that you have both the API Key and Secret Key provided by the CloudStack administrator for your account before proceeding with the signing process.

To show how to sign a request, we will re-use the previous example.

```

http://localhost:8080/client/api?command=deployVirtualMachine&serviceOfferingId=1&diskOfferingId=1&
templateId=2&zoneId=4&apiKey=miVr6X7u6bN_sdahOBpjNejPgEsT35eXq-jB8CG20YI3yaxXcgpyuaIRmFI_
EJTVwZ0nUkkJbPmY3y2bciKwFQ&signature=Lxx1DM40AjcXU%2FcaiK8RAP001hU%3D

```

Breaking this down, we have several distinct parts to this URL.

- Base URL : This is the base URL to the CloudStack Management Server.

```
http://localhost:8080
```

- API Path : This is the path to the API Servlet that processes the incoming requests.

```
/client/api?
```

- Command String : This part of the query string comprises of the command, its parameters, and the API Key that identifies the account.

Note : As with all query string parameters of field-value pairs, the “field”

component is case insensitive while all “value” values are case sensitive.

```
command=deployVirtualMachine&serviceOfferingId=1&diskOfferingId=1&templateId=2&zoneId=4&apiKey=miVr6X7u6bN_sdahO
jB8CG20YI3yaxXcgpyuaIRmFI_EJTVwZ0nUkkJbPmY3y2bciKwFQ
```

- Signature : This is the signature of the command string that is generated using a combination of the user’s Secret Key and the HMAC SHA-1 hashing algorithm.

```
&signature=Lxx1DM40AjcXU%2FcaiK8RAP001hU%3D
```

Every API request has the format Base URL+API Path+Command String+Signature.

To generate the signature.

1. For each field-value pair (as separated by a ‘&’) in the Command String, URL encode each value so that it can be safely sent via HTTP GET.

Note : Make sure all spaces are encoded as “%20” rather than “+”.

- Lower case the entire Command String and sort it alphabetically via the field for each field-value pair. The result of this step would look like the following.
- Take the sorted Command String and run it through the HMAC SHA-1 hashing algorithm (most programming languages offer a utility method to do this) with the user’s Secret Key. Base64 encode the resulting byte array in UTF-8 so that it can be safely transmitted via HTTP. The final string produced after Base64 encoding should be “Lxx1DM40AjcXU%2FcaiK8RAP001hU%3D”.

By reconstructing the final URL in the format Base URL+API Path+Command String+Signature, the final URL should look like :

```
:: http://localhost:8080/client/api?command=deployVirtualMachine&serviceOfferingId=
  1&diskOfferingId=1&templateId=2&zoneId=4&apiKey=miVr6X7u6bN_
  sdahOBpjNejPgEsT35eXq-jB8CG20YI3yaxXcgpyuaIRmFI_EJTVwZ0nUkkJbPmY3y2bciKwFQ&
  signature=Lxx1DM40AjcXU%2FcaiK8RAP001hU%3D
```

How to sign an API call with Python

To illustrate the procedure used to sign API calls we present a step by step interactive session using Python.

First import the required modules :

```
$python
Python 2.7.3 (default, Nov 17 2012, 19:54:34)
[GCC 4.2.1 Compatible Apple Clang 4.1 ((tags/Apples/clang-421.11.66))] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> import urllib
>>> import hashlib
>>> import hmac
>>> import base64
```

Define the endpoint of the Cloud, the command that you want to execute and the keys of the user.

```
>>> baseurl='http://localhost:8080/client/api?'
>>> request={}
>>> request['command']='listUsers'
>>> request['response']='json'
>>> request['apikey']='plgWJfZK4gyS3mOMTVmjUVg-X-jlWlnfaUJ9GAbBbf9EdM-
↳ kAYMmAiLqzzq1ElZLYq_u38zCm0bewzGUdP66mg'
>>> secretkey='VDaACYb0LV9eNjTetIOElcVQkvJck_J_Q1jX_
↳ FcHRj87ZKiy0z0ty0ZsYBkoXkY9b7eq1EhwJaw7FF3akA3KBQ'
```

Build the request string :

```
>>> request_str='&'.join(['=' .join([k,urllib.quote_plus(request[k])]) for k in_
↳ request.keys()])
>>> request_str
'apikey=plgWJfZK4gyS3mOMTVmjUVg-X-jlWlnfaUJ9GAbBbf9EdM-kAYMmAiLqzzq1ElZLYq_
↳ u38zCm0bewzGUdP66mg&command=listUsers&response=json'
```

Compute the signature with hmac, do a 64 bit encoding and a url encoding :

```
>>> sig_str='&'.join(['=' .join([k.lower(),urllib.quote_plus(request[k].lower()).
↳ replace('+','%20')]) for k in sorted(request.iterkeys())])
>>> sig_str 'apikey=plgwjfk4gys3momtmvmjuvg-x-jlwlfnfauj9gabbbf9edm-
↳ kaymmailqzzq1elzlyq_u38zcm0bewzgudp66mg&command=listusers&response=json'
```

```
>>> sig=hmac.new(secretkey,sig_str,hashlib.sha1)
>>> sig
<hmac.HMAC instance at 0x10d91d680>
>>> sig=hmac.new(secretkey,sig_str,hashlib.sha1).digest()
>>> sig
'M:]x0e\xaf\xfb\x8f\x2y\xflp\x9l\x1e\x89\x8a\xa1\x05\xc4A\xdb'
>>> sig=base64.encodestring(hmac.new(secretkey,sig_str,hashlib.sha1).digest())
>>> sig
'TTpdDq/7j/J58XCRHomKoQXEQds=\n'
>>> sig=base64.encodestring(hmac.new(secretkey,sig_str,hashlib.sha1).digest()).strip()
>>> sig
'TTpdDq/7j/J58XCRHomKoQXEQds='
>>> sig=urllib.quote_plus(base64.encodestring(hmac.new(secretkey,sig_str,hashlib.
↳ sha1).digest()).strip())
```

Finally, build the entire string and do an http GET :

```
>>> req=baseurl+request_str+'&signature='+sig
>>> req
'http://localhost:8080/client/api?apikey=plgWJfZK4gyS3mOMTVmjUVg-X-
↳ jlWlnfaUJ9GAbBbf9EdM-kAYMmAiLqzzq1ElZLYq_u38zCm0bewzGUdP66mg&command=listUsers&
↳ response=json&signature=TTpdDq%2F7j%2FJ58XCRHomKoQXEQds%3D'
>>> res=urllib2.urlopen(req)
>>> res.read()
'{"listusersresponse": {"count":3, "user": [ {"id":"7ed6d5da-93b2-4545-a502-
↳ 23d20b48ef2a", "username":"admin", "firstname":"admin", "lastname":"cloud", "created":
↳ "2012-07-05T12:18:27-0700", "state":"enabled", "account":"admin", "accounttype":1,
↳ "domainid":"8a111e58-e155-4482-93ce-84efff3c7c77", "domain":"ROOT", "apikey":
↳ "plgWJfZK4gyS3mOMTVmjUVg-X-jlWlnfaUJ9GAbBbf9EdM-kAYMmAiLqzzq1ElZLYq_
↳ u38zCm0bewzGUdP66mg", "secretkey":"VDAcYb0LV9eNjTetIOElcVQkvJck_J_Q1jX_
↳ FcHRj87ZKiy0z0ty0ZsYBkoXkY9b7eq1EhwJaw7FF3akA3KBQ", "accountid":"7548ac03-af1d-4c1c-
↳ 9064-2f3e2c0eda0d"}, {"id":"1fea6418-5576-4989-a21e-4790787bbee3", "username":"runseb
↳ ", "firstname":"foobar", "lastname":"goa", "email":"joe@smith.com", "created":"2013-04-
↳ 10T16:52:06-0700", "state":"enabled", "account":"admin", "accounttype":1, "domainid":
↳ "8a111e58-e155-4482-93ce-84efff3c7c77", "domain":"ROOT", "apikey":
↳ "Xhsb3MewjJQaXXMsZrCLvQI9_NPy_
↳ UcbDj1QXikKvBdC9MDSPwWdtZ1bUY1H7JBEYtTDDLY3yuchCeW778GkBA", "secretkey":
↳ "gIsgmi8C5YwxMHjX5o51pSe0kqs6JnKriw0jJBLceY5bgnfzKjL4aM6ctJX-
↳ i1ddQIHJLbLJDK9MRzsKk6xZ_w", "accountid":"7548ac03-af1d-4c1c-9064-2f3e2c0eda0d"}, {
↳ "id":"52f65396-183c-4473-883f-a37e7bb93967", "username":"toto", "firstname":"john",
↳ "lastname":"smith", "email":"john@smith.com", "created":"2013-04-23T04:27:22-0700",
↳ "state":"enabled", "account":"admin", "accounttype":1, "domainid":"8a111e58-e155-4482-
↳ 93ce-84efff3c7c77", "domain":"ROOT", "apikey":"THaA6fFWS_OmvU8od201omxFC8yKNL_
↳ Hc5ZCS77LFCJsRzSx48JyZucbUul6XYbEg-ZyXmL_wuEpECzK-wKnow", "secretkey":
↳ "O5ywpqJorAsEBKR_5jEvertGHfWL1Y_j1E4Z_
↳ iCr8OKCYcsPIOdVcfzjJQ8YqK0a5EzSpoRrjOFiLSG0hQrYnDA", "accountid":"7548ac03-af1d-4c1c-
↳ 9064-2f3e2c0eda0d"} ] } }'
```

Enabling API Call Expiration

You can set an expiry timestamp on API calls to prevent replay attacks over non-secure channels, such as HTTP. The server tracks the expiry timestamp you have specified and rejects all the subsequent API requests that come in after this validity period.

To enable this feature, add the following parameters to the API request :

- signatureVersion=3 : If the signatureVersion parameter is missing or is not equal to 3, the expires parameter is ignored in the API request.

- `expires=YYYY-MM-DDThh:mm:ssZ` : Specifies the date and time at which the signature included in the request is expired. The timestamp is expressed in the YYYY-MM-DDThh:mm:ssZ format, as specified in the ISO 8601 standard.

For example :

```
expires=2011-10-10T12:00:00+0530
```

A sample API request with expiration is given below :

```
http://<IPAddress>:8080/client/api?command=listZones&signatureVersion=3&expires=2011-10-10T12:00:00+0530&apiKey=miVr6X7u6bN_sdahOBpjNejPgEsT35eXq-jB8CG20YI3yaxXcgpyuaIRmFI_EJTVwZ0nUkkJbPmY3y2
```

Limiting the Rate of API Requests

You can limit the rate at which API requests can be placed for each account. This is useful to avoid malicious attacks on the Management Server, prevent performance degradation, and provide fairness to all accounts.

If the number of API calls exceeds the threshold, an error message is returned for any additional API calls. The caller will have to retry these API calls at another time.

Configuring the API Request Rate

To control the API request rate, use the following global configuration settings :

- *api.throttling.enabled* - Enable/Disable API throttling. By default, this setting is false, so API throttling is not enabled.
- *api.throttling.interval* (in seconds) - Time interval during which the number of API requests is to be counted. When the interval has passed, the API count is reset to 0.
- *api.throttling.max* - Maximum number of APIs that can be placed within the *api.throttling.interval* period.
- *api.throttling.cachesize* - Cache size for storing API counters. Use a value higher than the total number of accounts managed by the cloud. One cache entry is needed for each account, to store the running API total for that account.

Limitations on API Throttling

The following limitations exist in the current implementation of this feature.

Note : Even with these limitations, CloudStack is still able to effectively use

API throttling to avoid malicious attacks causing denial of service.

- In a deployment with multiple Management Servers, the cache is not synchronized across them. In this case, CloudStack might not be able to ensure that only the exact desired number of API requests are allowed. In the worst case, the number of API calls that might be allowed is (number of Management Servers) * (*api.throttling.max*).
- The API commands `resetApiLimit` and `getApiLimit` are limited to the Management Server where the API is invoked.

API Responses

Response Formats : XML and JSON

CloudStack supports two formats as the response to an API call. The default response is XML. If you would like the response to be in JSON, add `&response=json` to the Command String.

The two response formats differ in how they handle blank fields. In JSON, if there is no value for a response field, it will not appear in the response. If all the fields were empty, there might be no response at all. In XML, even if there is no value to be returned, an empty field will be returned as a placeholder XML element.

Sample XML Response :

```
<listipaddressesresponse>
  <allocatedipaddress>
    <ipaddress>192.168.10.141</ipaddress>
    <allocated>2009-09-18T13:16:10-0700</allocated>
    <zoneid>4</zoneid>
    <zonename>WC</zonename>
    <issourcenat>true</issourcenat>
  </allocatedipaddress>
</listipaddressesresponse>
```

Sample JSON Response :

```
{ "listipaddressesresponse" :
  { "allocatedipaddress" :
    [
      {
        "ipaddress" : "192.168.10.141",
        "allocated" : "2009-09-18T13:16:10-0700",
        "zoneid" : "4",
        "zonename" : "WC",
        "issourcenat" : "true"
      }
    ]
  }
}
```

Maximum Result Pages Returned

For each cloud, there is a default upper limit on the number of results that any API command will return in a single page. This is to help prevent overloading the cloud servers and prevent DOS attacks. For example, if the page size limit is 500 and a command returns 10,000 results, the command will return 20 pages.

The default page size limit can be different for each cloud. It is set in the global configuration parameter *default.page.size*. If your cloud has many users with lots of VMs, you might need to increase the value of this parameter. At the same time, be careful not to set it so high that your site can be taken down by an enormous return from an API call. For more information about how to set global configuration parameters, see “Describe Your Deployment” in the Installation Guide.

To decrease the page size limit for an individual API command, override the global setting with the page and pagesize parameters, which are available in any list* command (listCapabilities, listDiskOfferings, etc.).

- Both parameters must be specified together.
- The value of the pagesize parameter must be smaller than the value of default.page.size. That is, you can not increase the number of possible items in a result page, only decrease it.

For syntax information on the list* commands, see the API Reference.

Error Handling

If an error occurs while processing an API request, the appropriate response in the format specified is returned. Each error response consists of an error code and an error text describing what possibly can go wrong. Below is a list of possible error codes :

You can now find the CloudStack-specific error code in the exception response for each type of exception. The following list of error codes is added to the new class named `CSExceptionErrorCode`.

4250 : “com.cloud.utils.exception.CloudRuntimeException”
4255 : “com.cloud.utils.exception.ExceptionUtil”
4260 : “com.cloud.utils.exception.ExecutionException”
4265 : “com.cloud.utils.exception.HypervisorVersionChangedException”
4270 : “com.cloud.utils.exception.RuntimeCloudException”
4275 : “com.cloud.exception.CloudException”
4280 : “com.cloud.exception.AccountLimitException”
4285 : “com.cloud.exception.AgentUnavailableException”
4290 : “com.cloud.exception.CloudAuthenticationException”
4295 : “com.cloud.exception.CloudExecutionException”
4300 : “com.cloud.exception.ConcurrentOperationException”
4305 : “com.cloud.exception.ConflictingNetworkSettingsException”
4310 : “com.cloud.exception.DiscoveredWithErrorException”
4315 : “com.cloud.exception.HAStateException”
4320 : “com.cloud.exception.InsufficientAddressCapacityException”
4325 : “com.cloud.exception.InsufficientCapacityException”
4330 : “com.cloud.exception.InsufficientNetworkCapacityException”
4335 : “com.cloud.exception.InsufficientServerCapacityException”
4340 : “com.cloud.exception.InsufficientStorageCapacityException”
4345 : “com.cloud.exception.InternalErrorException”
4350 : “com.cloud.exception.InvalidParameterValueException”
4355 : “com.cloud.exception.ManagementServerException”
4360 : “com.cloud.exception.NetworkRuleConflictException”
4365 : “com.cloud.exception.PermissionDeniedException”
4370 : “com.cloud.exception.ResourceAllocationException”
4375 : “com.cloud.exception.ResourceInUseException”
4380 : “com.cloud.exception.ResourceUnavailableException”
4385 : “com.cloud.exception.StorageUnavailableException”
4390 : “com.cloud.exception.UnsupportedServiceException”
4395 : “com.cloud.exception.VirtualMachineMigrationException”
4400 : “com.cloud.exception.AccountLimitException”

4405 : "com.cloud.exception.AgentUnavailableException"
 4410 : "com.cloud.exception.CloudAuthenticationException"
 4415 : "com.cloud.exception.CloudException"
 4420 : "com.cloud.exception.CloudExecutionException"
 4425 : "com.cloud.exception.ConcurrentOperationException"
 4430 : "com.cloud.exception.ConflictingNetworkSettingsException"
 4435 : "com.cloud.exception.ConnectionException"
 4440 : "com.cloud.exception.DiscoveredWithErrorException"
 4445 : "com.cloud.exception.DiscoveryException"
 4450 : "com.cloud.exception.HAStateException"
 4455 : "com.cloud.exception.InsufficientAddressCapacityException"
 4460 : "com.cloud.exception.InsufficientCapacityException"
 4465 : "com.cloud.exception.InsufficientNetworkCapacityException"
 4470 : "com.cloud.exception.InsufficientServerCapacityException"
 4475 : "com.cloud.exception.InsufficientStorageCapacityException"
 4480 : "com.cloud.exception.InsufficientVirtualNetworkCapacityException"
 4485 : "com.cloud.exception.InternalErrorException"
 4490 : "com.cloud.exception.InvalidParameterValueException"
 4495 : "com.cloud.exception.ManagementServerException"
 4500 : "com.cloud.exception.NetworkRuleConflictException"
 4505 : "com.cloud.exception.PermissionDeniedException"
 4510 : "com.cloud.exception.ResourceAllocationException"
 4515 : "com.cloud.exception.ResourceInUseException"
 4520 : "com.cloud.exception.ResourceUnavailableException"
 4525 : "com.cloud.exception.StorageUnavailableException"
 4530 : "com.cloud.exception.UnsupportedServiceException"
 4535 : "com.cloud.exception.VirtualMachineMigrationException"
 9999 : "org.apache.cloudstack.api.ServerApiException"

An HTTP error code of 401 is always returned if API request was rejected due to bad signatures, missing API Keys, or the user simply did not have the permissions to execute the command.

Asynchronous Commands

Asynchronous commands were introduced in CloudStack 2.x. Commands are designated as asynchronous when they can potentially take a long period of time to complete such as creating a snapshot or disk volume. They differ from synchronous commands by the following :

- They are identified in the API Reference by an (A).
- They will immediately return a job ID to refer to the job that will be responsible in processing the command.

- If executed as a “create” resource command, it will return the resource ID as well as the job ID. You can periodically check the status of the job by making a simple API call to the command, *queryAsyncJobResult* and passing in the job ID.

Job Status

The key to using an asynchronous command is the job ID that is returned immediately once the command has been executed. With the job ID, you can periodically check the job status by making calls to *queryAsyncJobResult* command. The command will return three possible job status integer values :

- 0 - Job is still in progress. Continue to periodically poll for any status changes.
- 1 - Job has successfully completed. The job will return any successful response values associated with command that was originally executed.
- 2 - Job has failed to complete. Please check the “jobresultcode” tag for failure reason code and “jobresult” for the failure reason.

Example

The following shows an example of using an asynchronous command. Assume the API command :

```
command=deployVirtualMachine&zoneId=1&serviceOfferingId=1&diskOfferingId=1&
↳templateId=1
```

CloudStack will immediately return a job ID and any other additional data.

```
<deployvirtualmachineresponse>
  <jobid>1</jobid>
  <id>100</id>
</deployvirtualmachineresponse>
```

Using the job ID, you can periodically poll for the results by using the *queryAsyncJobResult* command.

```
command=queryAsyncJobResult&jobId=1
```

Three possible results could come from this query.

Job is still pending :

```
<queryasyncjobresult>
  <jobid>1</jobid>
  <jobstatus>0</jobstatus>
  <jobprocstatus>1</jobprocstatus>
</queryasyncjobresult>
```

Job has succeeded :

```
<queryasyncjobresultresponse cloud-stack-version="3.0.1.6">
  <jobid>1</jobid>
  <jobstatus>1</jobstatus>
  <jobprocstatus>0</jobprocstatus>
  <jobresultcode>0</jobresultcode>
  <jobresulttype>object</jobresulttype>
  <jobresult>
    <virtualmachine>
      <id>450</id>
      <name>i-2-450-VM</name>
      <displayname>i-2-450-VM</displayname>
```



```

<account>admin</account>
<domainid>1</domainid>
<domain>ROOT</domain>
<created>2011-03-10T18:20:25-0800</created>
<state>Running</state>
<haenable>>false</haenable>
<zoneid>1</zoneid>
<zonenname>San Jose 1</zonenname>
<hostid>2</hostid>
<hostname>905-13.sjc.lab.vmops.com</hostname>
<templateid>1</templateid>
<templatename>CentOS 5.3 64bit LAMP</templatename>
<templatedisplaytext>CentOS 5.3 64bit LAMP</templatedisplaytext>
<passwordenabled>>false</passwordenabled>
<serviceofferingid>1</serviceofferingid>
<serviceofferingname>Small Instance</serviceofferingname>
<cpunumber>1</cpunumber>
<cpuspeed>500</cpuspeed>
<memory>512</memory>
<guestosid>12</guestosid>
<rootdeviceid>0</rootdeviceid>
<rootdevicetype>NetworkFilesystem</rootdevicetype>
<nic>
  <id>561</id>
  <networkid>205</networkid>
  <netmask>255.255.255.0</netmask>
  <gateway>10.1.1.1</gateway>
  <ipaddress>10.1.1.225</ipaddress>
  <isolationuri>vlan://295</isolationuri>
  <broadcasturi>vlan://295</broadcasturi>
  <traffictype>Guest</traffictype>
  <type>Virtual</type>
  <isdefault>true</isdefault>
</nic>
<hypervisor>XenServer</hypervisor>
</virtualmachine>
</jobresult>
</queryasyncjobresultresponse>

```

Job has failed :

```

<queryasyncjobresult>
  <jobid>1</jobid>
  <jobstatus>2</jobstatus>
  <jobprocstatus>0</jobprocstatus>
  <jobresultcode>551</jobresultcode>
  <jobresultttype>text</jobresultttype>
  <jobresult>Unable to deploy virtual machine id = 100 due to not enough capacity
  ↪</jobresult>
</queryasyncjobresult>

```

Event Types

Types	Events
VM	VM.CREATE VM.DESTROY VM.START VM.STOP VM.REBOOT VM.UPDATE VM.UPGRADE VM.DYNAMIC.SCALE VM.RESETPASSWORD VM.RESETSSHKEY VM.MIGRATE VM.MOVE VM.RESTORE
Domain Router	ROUTER.CREATE ROUTER.DESTROY ROUTER.START ROUTER.STOP ROUTER.REBOOT ROUTER.HA ROUTER.UPGRADE
Console proxy	PROXY.CREATE PROXY.DESTROY PROXY.START PROXY.STOP PROXY.REBOOT PROXY.HA
VNC Console Events	VNC.CONNECT VNC.DISCONNECT
Network Events	NET.IPASSIGN NET.IPRELEASE PORTABLE.IPASSIGN PORTABLE.IPRELEASE NET.RULEADD NET.RULEDELETE NET.RULEMODIFY NETWORK.CREATE NETWORK.DELETE NETWORK.UPDATE FIREWALL.OPEN FIREWALL.CLOSE
NIC Events	NIC.CREATE NIC.DELETE NIC.UPDATE NIC.DETAIL.ADD NIC.DETAIL.UPDATE NIC.DETAIL.REMOVE
Suite sur la page suivante	

Tableau 4.1 – Suite de la page précédente

Types	Events
Load Balancers	LB.ASSIGN.TO.RULE LB.REMOVE.FROM.RULE LB.CREATE LB.DELETE LB.STICKINESSPOLICY.CREATE LB.STICKINESSPOLICY.DELETE LB.HEALTHCHECKPOLICY.CREATE LB.HEALTHCHECKPOLICY.DELETE LB.UPDATE
Global Load Balancer rules	GLOBAL.LB.ASSIGN GLOBAL.LB.REMOVE GLOBAL.LB.CREATE GLOBAL.LB.DELETE GLOBAL.LB.UPDATE
Account events	ACCOUNT.ENABLE ACCOUNT.DISABLE ACCOUNT.CREATE ACCOUNT.DELETE ACCOUNT.UPDATE ACCOUNT.MARK.DEFAULT.ZONE
UserVO Events	USER.LOGIN USER.LOGOUT USER.CREATE USER.DELETE USER.DISABLE USER.UPDATE USER.ENABLE USER.LOCK
Registering SSH keypair events	REGISTER.SSH.KEYPAIR
Register for user API and secret keys	REGISTER.USER.KEY
Template Events	TEMPLATE.CREATE TEMPLATE.DELETE TEMPLATE.UPDATE TEMPLATE.DOWNLOAD.START TEMPLATE.DOWNLOAD.SUCCESS TEMPLATE.DOWNLOAD.FAILED TEMPLATE.COPY TEMPLATE.EXTRACT TEMPLATE.UPLOAD TEMPLATE.CLEANUP
Suite sur la page suivante	

Tableau 4.1 – Suite de la page précédente

Types	Events
Volume Events	VOLUME.CREATE VOLUME.DELETE VOLUME.ATTACH VOLUME.DETACH VOLUME.EXTRACT VOLUME.UPLOAD VOLUME.MIGRATE VOLUME.RESIZE VOLUME.DETAIL.UPDATE VOLUME.DETAIL.ADD VOLUME.DETAIL.REMOVE
Domains	DOMAIN.CREATE DOMAIN.DELETE DOMAIN.UPDATE
Snapshots	SNAPSHOT.CREATE SNAPSHOT.DELETE SNAPSHOTPOLICY.CREATE SNAPSHOTPOLICY.UPDATE SNAPSHOTPOLICY.DELETE
ISO	ISO.CREATE ISO.DELETE ISO.COPY ISO.ATTACH ISO.DETACH ISO.EXTRACT ISO.UPLOAD
SSVM	SSVM.CREATE SSVM.DESTROY SSVM.START SSVM.STOP SSVM.REBOOT SSVM.HA
Service Offerings	SERVICE.OFFERING.CREATE SERVICE.OFFERING.EDIT SERVICE.OFFERING.DELETE
Disk Offerings	DISK.OFFERING.CREATE DISK.OFFERING.EDIT DISK.OFFERING.DELETE
Network offerings	NETWORK.OFFERING.CREATE NETWORK.OFFERING.ASSIGN NETWORK.OFFERING.EDIT NETWORK.OFFERING.REMOVE NETWORK.OFFERING.DELETE
Pods	POD.CREATE POD.EDIT POD.DELETE
Zones	ZONE.CREATE ZONE.EDIT ZONE.DELETE
Suite sur la page suivante	

Tableau 4.1 – Suite de la page précédente

Types	Events
VLANs/IP ranges	VLAN.IP.RANGE.CREATE VLAN.IP.RANGE.DELETE VLAN.IP.RANGE.DEDICATE VLAN.IP.RANGE.RELEASE STORAGE.IP.RANGE.CREATE STORAGE.IP.RANGE.DELETE STORAGE.IP.RANGE.UPDATE
Configuration Table	CONFIGURATION.VALUE.EDIT
Security Groups	SG.AUTH.INGRESS SG.REVOKE.INGRESS SG.AUTH.EGRESS SG.REVOKE.EGRESS SG.CREATE SG.DELETE SG.ASSIGN SG.REMOVE
Host	HOST.RECONNECT
Maintenance	MAINT.CANCEL MAINT.CANCEL.PS MAINT.PREPARE MAINT.PREPARE.PS
VPN	VPN.REMOTE.ACCESS.CREATE VPN.REMOTE.ACCESS.DESTROY VPN.USER.ADD VPN.USER.REMOVE VPN.S2S.VPN.GATEWAY.CREATE VPN.S2S.VPN.GATEWAY.DELETE VPN.S2S.CUSTOMER.GATEWAY.CREATE VPN.S2S.CUSTOMER.GATEWAY.DELETE VPN.S2S.CUSTOMER.GATEWAY.UPDATE VPN.S2S.CONNECTION.CREATE VPN.S2S.CONNECTION.DELETE VPN.S2S.CONNECTION.RESET
Network	NETWORK.RESTART
Custom certificates	UPLOAD.CUSTOM.CERTIFICATE
OneToOnenat	STATICNAT.ENABLE STATICNAT.DISABLE ZONE.VLAN.ASSIGN ZONE.VLAN.RELEASE
Projects	PROJECT.CREATE PROJECT.UPDATE PROJECT.DELETE PROJECT.ACTIVATE PROJECT.SUSPEND PROJECT.ACCOUNT.ADD PROJECT.INVITATION.UPDATE PROJECT.INVITATION.REMOVE PROJECT.ACCOUNT.REMOVE
Network as a Service	NETWORK.ELEMENT.CONFIGURE
Suite sur la page suivante	

Tableau 4.1 – Suite de la page précédente

Types	Events
Physical Network Events	PHYSICAL.NETWORK.CREATE PHYSICAL.NETWORK.DELETE PHYSICAL.NETWORK.UPDATE
Physical Network Service Provider Events	SERVICE.PROVIDER.CREATE SERVICE.PROVIDER.DELETE SERVICE.PROVIDER.UPDATE
Physical Network Traffic Type Events	TRAFFIC.TYPE.CREATE TRAFFIC.TYPE.DELETE TRAFFIC.TYPE.UPDATE
External network device events	PHYSICAL.LOADBALANCER.ADD PHYSICAL.LOADBALANCER.DELETE PHYSICAL.LOADBALANCER.CONFIGURE
External switch management device events For example : Cisco Nexus 1000v Virtual Supervisor Module.	SWITCH.MGMT.ADD SWITCH.MGMT.DELETE SWITCH.MGMT.CONFIGURE SWITCH.MGMT.ENABLE SWITCH.MGMT.DISABLE PHYSICAL.FIREWALL.ADD PHYSICAL.FIREWALL.DELETE PHYSICAL.FIREWALL.CONFIGURE
VPC	VPC.CREATE VPC.UPDATE VPC.DELETE VPC.RESTART
Network ACL	NETWORK.ACL.CREATE NETWORK.ACL.DELETE NETWORK.ACL.REPLACE NETWORK.ACL.ITEM.CREATE NETWORK.ACL.ITEM.UPDATE NETWORK.ACL.ITEM.DELETE
VPC offerings	VPC.OFFERING.CREATE VPC.OFFERING.UPDATE VPC.OFFERING.DELETE
Private gateway	PRIVATE.GATEWAY.CREATE PRIVATE.GATEWAY.DELETE
Static routes	STATIC.ROUTE.CREATE STATIC.ROUTE.DELETE
Tag-related events	CREATE_TAGS DELETE_TAGS
Meta data-related events	CREATE_RESOURCE_DETAILS DELETE_RESOURCE_DETAILS
VM snapshot events	VMSNAPSHOT.CREATE VMSNAPSHOT.DELETE VMSNAPSHOT.REVERTTO
External network device events	PHYSICAL.NVPCONTROLLER.ADD PHYSICAL.NVPCONTROLLER.DELETE PHYSICAL.NVPCONTROLLER.CONFIGURE
Suite sur la page suivante	

Tableau 4.1 – Suite de la page précédente

Types	Events
AutoScale	COUNTER.CREATE COUNTER.DELETE CONDITION.CREATE CONDITION.DELETE AUTOSCALEPOLICY.CREATE AUTOSCALEPOLICY.UPDATE AUTOSCALEPOLICY.DELETE AUTOSCALEVMPROFILE.CREATE AUTOSCALEVMPROFILE.DELETE AUTOSCALEVMPROFILE.UPDATE AUTOSCALEVMGROUP.CREATE AUTOSCALEVMGROUP.DELETE AUTOSCALEVMGROUP.UPDATE AUTOSCALEVMGROUP.ENABLE AUTOSCALEVMGROUP.DISABLE PHYSICAL.DHCP.ADD PHYSICAL.DHCP.DELETE PHYSICAL.PXE.ADD PHYSICAL.PXE.DELETE AG.CREATE AG.DELETE AG.ASSIGN AG.REMOVE VM.AG.UPDATE INTERNALLBVM.START INTERNALLBVM.STOP HOST.RESERVATION.RELEASE
Dedicated guest vlan range	GUESTVLANRANGE.DEDICATE GUESTVLANRANGE.RELEASE PORTABLE.IP.RANGE.CREATE PORTABLE.IP.RANGE.DELETE PORTABLE.IP.TRANSFER
Dedicated Resources	DEDICATE.RESOURCE DEDICATE.RESOURCE.RELEASE VM.RESERVATION.CLEANUP UCS.ASSOCIATEPROFILE UCS.DISASSOCIATEPROFILE

Time Zones

The following time zone identifiers are accepted by PRODUCT. There are several places that have a time zone as a required or optional parameter. These include scheduling recurring snapshots, creating a user, and specifying the usage time zone in the Configuration table.

Etc/GMT+12	Etc/GMT+11	Pacific/Samoa
Pacific/Honolulu	US/Alaska	America/Los_Angeles
Mexico/BajaNorte	US/Arizona	US/Mountain
America/Chihuahua	America/Chicago	America/Costa_Rica
America/Mexico_City	Canada/Saskatchewan	America/Bogota
America/New_York	America/Caracas	America/Asuncion
America/Cuiaba	America/Halifax	America/La_Paz
America/Santiago	America/St_Johns	America/Araguaina
America/Argentina/Buenos_Aires	America/Cayenne	America/Godthab
America/Montevideo	Etc/GMT+2	Atlantic/Azores
Atlantic/Cape_Verde	Africa/Casablanca	Etc/UTC
Atlantic/Reykjavik	Europe/London	CET
Europe/Bucharest	Africa/Johannesburg	Asia/Beirut
Africa/Cairo	Asia/Jerusalem	Europe/Minsk
Europe/Moscow	Africa/Nairobi	Asia/Karachi
Asia/Kolkata	Asia/Bangkok	Asia/Shanghai
Asia/Kuala_Lumpur	Australia/Perth	Asia/Taipei
Asia/Tokyo	Asia/Seoul	Australia/Adelaide
Australia/Darwin	Australia/Brisbane	Australia/Canberra
Pacific/Guam	Pacific/Auckland	

Plugins

Storage Plugins

This section gives an outline of how to implement a plugin to integrate a third-party storage provider. For details and an example, you will need to read the code.

Note : Example code is available at : [plugins/storage/volume/sample](#)

Third party storage providers can integrate with CloudStack to provide either primary storage or secondary storage. For example, CloudStack provides plugins for Amazon Simple Storage Service (S3) or OpenStack Object Storage (Swift). The S3 plugin can be used for any object storage that supports the Amazon S3 interface.

Additional third party object storages that do not support the S3 interface can be integrated with CloudStack by writing plugin software that uses the object storage plugin framework. Several new interfaces are available so that storage providers can develop vendor-specific plugins based on well-defined contracts that can be seamlessly managed by CloudStack.

Artifacts such as templates, ISOs and snapshots are kept in storage which CloudStack refers to as secondary storage. To improve scalability and performance, as when a number of hosts access secondary storage concurrently, object storage can be used for secondary storage. Object storage can also provide built-in high availability capability. When using object storage, access to secondary storage data can be made available across multiple zones in a region. This is a huge benefit, as it is no longer necessary to copy templates, snapshots etc. across zones as would be needed in an environment using only zone-based NFS storage.

The user enables a storage plugin through the UI. A new dialog box choice is offered to select the storage provider. Depending on which provider is selected, additional input fields may appear so that the user can provide the additional details required by that provider, such as a user name and password for a third-party storage account.

Overview of How to Write a Storage Plugin

To add a third-party storage option to CloudStack, follow these general steps (explained in more detail later in this section) :

1. Implement the following interfaces in Java :
 - `DataStoreDriver`
 - `DataStoreLifecycle`
 - `DataStoreProvider`
 - `VMSnapshotStrategy` (if you want to customize the VM snapshot functionality)
2. Hardcode your plugin's required additional input fields into the code for the Add Secondary Storage or Add Primary Storage dialog box.
3. Place your .jar file in *plugins/storage/volume/* or *plugins/storage/image/*.
4. Edit */client/tomcatconf/componentContext.xml.in*.
5. Edit *client/pom.xml*.

Implementing `DataStoreDriver`

`DataStoreDriver` contains the code that CloudStack will use to provision the object store, when needed.

You must implement the following methods :

- `getTO()`
- `getStoreTO()`
- `createAsync()`
- `deleteAsync()`

The following methods are optional :

- `resize()`
- `canCopy()` is optional. If you set it to true, then you must implement `copyAsync()`.

Implementing `DataStoreLifecycle`

`DataStoreLifecycle` contains the code to manage the storage operations for ongoing use of the storage. Several operations are needed, like create, maintenance mode, delete, etc.

You must implement the following methods :

- `initialize()`
- `maintain()`
- `cancelMaintain()`
- `deleteDataStore()`
- Implement one of the `attach*()` methods depending on what scope you want the storage to have : `attachHost()`, `attachCluster()`, or `attachZone()`.

Implementing `DataStoreProvider`

`DataStoreProvider` contains the main code of the data store.

You must implement the following methods :

- `getDatastoreLifeCycle()`
- `getDataStoreDriver()`
- `getTypes()`. Returns one or more types of storage for which this data store provider can be used. For secondary object storage, return `IMAGE`, and for a Secondary Staging Store, return `ImageCache`.
- `configure()`. First initialize the lifecycle implementation and the driver implementation, then call `registerDriver()` to register the new object store provider instance with CloudStack.

- getName(). Returns the unique name of your provider ; for example, this can be used to get the name to display in the UI.

The following methods are optional :

- getHostListener() is optional ; it's for monitoring the status of the host.

Implementing VMSnapshotStrategy

VMSnapshotStrategy has the following methods :

- takeVMSnapshot()
- deleteVMSnapshot()
- revertVMSnapshot()
- canHandle(). For a given VM snapshot, tells whether this implementation of VMSnapshotStrategy can handle it.

Place the .jar File in the Right Directory

For a secondary storage plugin, place your .jar file here :

```
plugins/storage/image/
```

For a primary storage plugin, place your .jar file here :

```
plugins/storage/volume/
```

Edit Configuration Files

First, edit the following file tell CloudStack to include your .jar file. Add a line to this file to tell the CloudStack Management Server that it now has a dependency on your code :

```
client/pom.xml
```

Place some facts about your code in the following file so CloudStack can run it :

```
/client/tomcatconf/componentContext.xml.in
```

In the section “Deployment configurations of various adapters,” add this :

```
<bean id="some unique ID" class="package name of your implementation of_  
↳DataStoreProvider"></bean>
```

In the section “Storage Providers,” add this :

```
<property name="providers">  
  <ref local="same ID from the bean tag's id attribute">  
</property>
```

Minimum Required Interfaces

The classes, interfaces, and methods used by CloudStack from the Amazon Web Services (AWS) Java SDK are listed in this section. An object storage that supports the S3 interface is minimally required to support the below in order to be compatible with CloudStack.

Interface AmazonS3

<http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/s3/AmazonS3.html>

Modifier and Type	Method and Description
Bucket	<code>createBucket(String bucketName)</code> Creates a new Amazon S3 bucket with the specified name in the default (US) region, <code>Region.US_Standard</code> .
void	<code>deleteObject(String bucketName, String key)</code> Deletes the specified object in the specified bucket.
ObjectMetadata	<code>getObject(GetObjectRequest getObjectRequest, File destinationFile)</code> Gets the object metadata for the object stored in Amazon S3 under the specified bucket and key, and saves the object contents to the specified file.
S3Object	<code>getObject(String bucketName, String key)</code> Gets the object stored in Amazon S3 under the specified bucket and key.
URL	<code>generatePresignedUrl(String bucketName, String key, Date expiration, HttpMethod method)</code> Returns a pre-signed URL for accessing an Amazon S3 resource.
void	<code>deleteBucket(String bucketName)</code> Deletes the specified bucket.
List<Bucket>	<code>listBuckets()</code> Returns a list of all Amazon S3 buckets that the authenticated sender of the request owns.
ObjectListing	<code>listObjects(String bucketName, String prefix)</code> Returns a list of summary information about the objects in the specified bucket.
PutObjectResult	<code>putObject(PutObjectRequest putObjectRequest)</code> Uploads a new object to the specified Amazon S3 bucket.
PutObjectResult	<code>putObject(String bucketName, String key, File file)</code> Uploads the specified file to Amazon S3 under the specified bucket and key name.
PutObjectResult	<code>putObject(String bucketName, String key, InputStream input, ObjectMetadata metadata)</code> Uploads the specified input stream and object metadata to Amazon S3 under the specified bucket and key name.
void	<code>setEndpoint(String endpoint)</code> Overrides the default endpoint for this client.
void	<code>setObjectAcl(String bucketName, String key, CannedAccessControlList acl)</code> Sets the <code>CannedAccessControlList</code> for the specified object in Amazon S3 using one of the pre-configured <code>CannedAccessControlLists</code> .

Class TransferManager

<http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/s3/transfer/TransferManager.html>

Modifier and Type	Method and Description
Upload	upload(PutObjectRequest putObjectRequest) Schedules a new transfer to upload data to Amazon S3.

Class PutObjectRequest

<http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/services/s3/model/PutObjectRequest.html>

Modifier and Type	Method and Description
Upload	upload(PutObjectRequest putObjectRequest) Schedules a new transfer to upload data to Amazon S3.

Third Party UI Plugins

Using the new third-party plugin framework, you can write and install extensions to CloudStack. The installed and enabled plugins will appear in the UI alongside the other features. The code for the plugin is simply placed in a special directory within CloudStack's installed code at any time after CloudStack installation. The new plugin appears only when it is enabled by the cloud administrator.



The left navigation bar of the CloudStack UI has a new Plugins button to help you work with UI plugins.

How to Write a Plugin : Overview

The basic procedure for writing a plugin is :

1. Write the code and create the other files needed. You will need the plugin code itself (in Javascript), a thumbnail image, the plugin listing, and a CSS file.

All UI plugins have the following set of files :

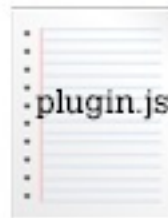
```

+-- cloudstack/
  +-- ui/
    +-- plugins/
      +-- csMyFirstPlugin/
        +-- config.js      --> Plugin metadata (title, author, vendor URL,
        etc.)
        +-- icon.png       --> Icon, shown on side nav bar and plugin
        listing
                                (should be square, and ~50x50px)
        +-- csMyFirstPlugin.css --> CSS file, loaded automatically when plugin
        loads
        +-- csMyFirstPlugin.js --> Main JS file, containing plugin code

```

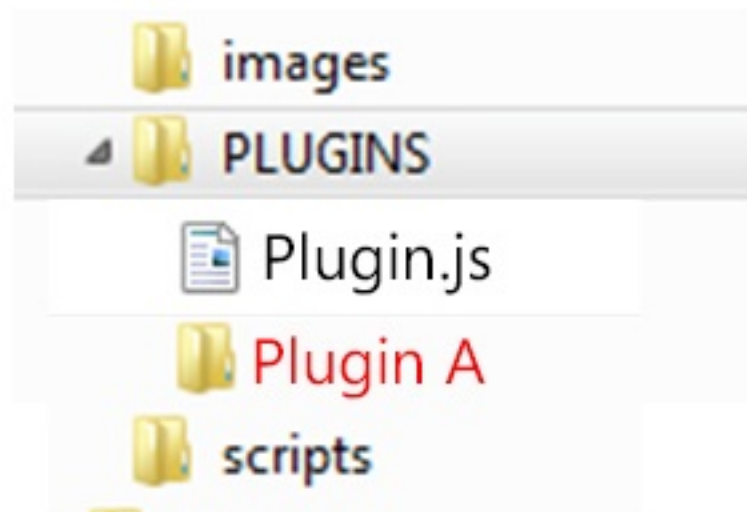
The same files must also be present at `/tomcat/webapps/client/plugins`.

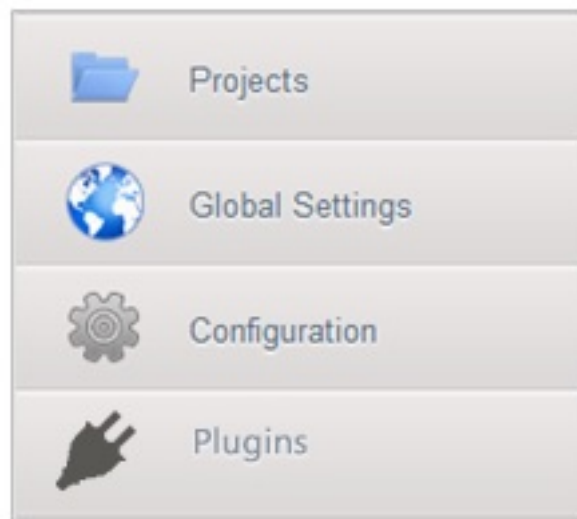
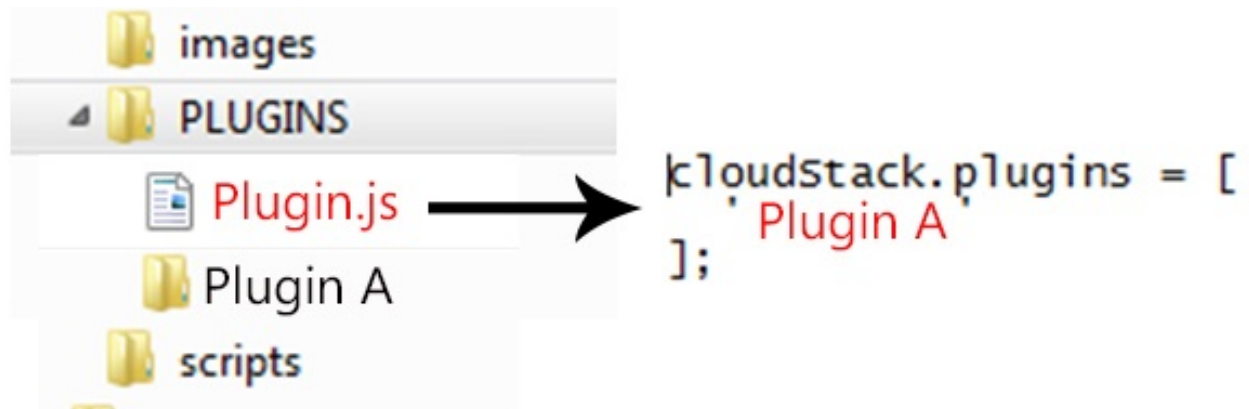
2. The CloudStack administrator adds the folder containing your plugin code under the CloudStack PLUGINS folder.
3. The administrator also adds the name of your plugin to the plugin.js file in the PLUGINS folder.
4. The next time the user refreshes the UI in the browser, your plugin will appear in the left navigation bar.



File Hierarchy

- plugin
- thumbnail
- description
- etc...





How to Write a Plugin : Implementation Details

This section requires an understanding of JavaScript and the CloudStack API. You don't need knowledge of specific frameworks for this tutorial (jQuery, etc.), since the CloudStack UI handles the front-end rendering for you.

There is much more to the CloudStack UI framework than can be described here. The UI is very flexible to handle many use cases, so there are countless options and variations. The best reference right now is to read the existing code for the main UI, which is in the /ui folder. Plugins are written in a very similar way to the main UI.

1. Create the directory to hold your plugin.

All plugins are composed of set of required files in the directory /ui/plugins/pluginID, where pluginID is a short name for your plugin. It's recommended that you prefix your folder name (for example, bfMyPlugin) to avoid naming conflicts with other people's plugins.

In this example, the plugin is named csMyFirstPlugin.

```
$ cd cloudstack/ui/plugins
$ mkdir csMyFirstPlugin
$ ls -l

total 8
drwxr-xr-x  2 bgregory  staff   68 Feb 11 14:44 csMyFirstPlugin
-rw-r--r--  1 bgregory  staff  101 Feb 11 14:26 plugins.js
```

2. Change to your new plugin directory.

```
$ cd csMyFirstPlugin
```

3. Set up the listing.

Add the file *config.js*, using your favorite editor.

```
$ vi config.js
```

Add the following content to config.js. This information will be displayed on the plugin listing page in the UI :

```
(function (cloudStack) {
  cloudStack.plugins.csMyFirstPlugin.config = {
    title: 'My first plugin',
    desc: 'Tutorial plugin',
    externalLink: 'http://www.cloudstack.org/',
    authorName: 'Test Plugin Developer',
    authorEmail: 'plugin.developer@example.com'
  };
}(cloudStack));
```

4. Add a new main section.

Add the file csMyFirstPlugin.js, using your favorite editor.

```
$ vi csMyFirstPlugin.js
```

Add the following content to csMyFirstPlugin.js :

```
(function (cloudStack) {
  cloudStack.plugins.csMyFirstPlugin = function(plugin) {
    plugin.ui.addSection({
      id: 'csMyFirstPlugin',
      title: 'My Plugin',
      preFilter: function(args) {
        return isAdmin();
      },
    },
```

```

    show: function() {
        return $('<div>').html('Content will go here');
    }
  });
};
}(cloudStack));

```

5. Register the plugin.

You now have the minimal content needed to run the plugin, so you can activate the plugin in the UI by adding it to `plugins.js`. First, edit the file :

```

$ cd cloudstack/ui/plugins
$ vi plugins.js

```

Now add the following to `plugins.js` :

```

(function($, cloudStack) {
  cloudStack.plugins = [
    'csMyFirstPlugin'
  ];
}(jQuery, cloudStack));

```

6. Check the plugin in the UI.

First, copy all the plugin code that you have created so far to `/tomcat/webapps/client/plugins`. Then refresh the browser and click Plugins in the side navigation bar. You should see your new plugin.

7. Make the plugin do something.

Right now, you just have placeholder content in the new plugin. It's time to add real code. In this example, you will write a basic list view, which renders data from an API call. You will list all virtual machines owned by the logged-in user. To do this, replace the 'show' function in the plugin code with a 'listView' block, containing the required syntax for a list view. To get the data, use the `listVirtualMachines` API call. Without any parameters, it will return VMs only for your active user. Use the provided 'apiCall' helper method to handle the server call. Of course, you are free to use any other method for making the AJAX call (for example, jQuery's `$.ajax` method).

First, open your plugin's JavaScript source file in your favorite editor :

```

$ cd csMyFirstPlugin
$ vi csMyFirstPlugin.js

```

Add the following code in `csMyFirstPlugin.js` :

```

(function (cloudStack) {
  cloudStack.plugins.csMyFirstPlugin = function(plugin) {
    plugin.ui.addSection({
      id: 'csMyFirstPlugin',
      title: 'My Plugin',
      preFilter: function(args) {
        return isAdmin();
      },

      // Render page as a list view
      listView: {
        id: 'testPluginInstances',
        fields: {
          name: { label: 'label.name' },
          instanceName: { label: 'label.internal.name' },
          displayName: { label: 'label.display.name' },
          zoneName: { label: 'label.zone.name' }
        }
      }
    });
  };
}(cloudStack));

```



```

    },
    dataProvider: function(args) {
        // API calls go here, to retrieve the data asynchronously
        //
        // On successful retrieval, call
        // args.response.success({ data: [data array] });
        plugin.ui.apiCall('listVirtualMachines', {
            success: function(json) {
                var vms = json.listvirtualmachinesresponse.virtualmachine;

                args.response.success({ data: vms });
            },
            error: function(errorMessage) {
                args.response.error(errorMessage)
            }
        });
    }
}
});
}(cloudStack));

```

8. Test the plugin.

First, copy all the plugin code that you have created so far to `/tomcat/webapps/client/plugins`. Then refresh the browser. You can see that your placeholder content was replaced with a list table, containing 4 columns of virtual machine data.

9. Add an action button.

Let's add an action button to the list view, which will reboot the VM. To do this, add an actions block under `listView`. After specifying the correct format, the actions will appear automatically to the right of each row of data.

```
$ vi csMyFirstPlugin.js
```

Now add the following new code in `csMyFirstPlugin.js`. (The dots ... show where we have omitted some existing code for the sake of space. Don't actually cut and paste that part) :

```

...
listView: {
    id: 'testPluginInstances',
    ...

    actions: {
        // The key/ID you specify here will determine what icon is
        // shown in the UI for this action,
        // and will be added as a CSS class to the action's element
        // (i.e., '.action.restart')
        //
        // -- here, 'restart' is a predefined name in CloudStack that will
        // automatically show a 'reboot' arrow as an icon;
        // this can be changed in csMyFirstPlugin.css
        restart: {
            label: 'Restart VM',
            messages: {
                confirm: function() { return 'Are you sure you want to restart_
↵this VM?' },
                notification: function() { return 'Rebooted VM' }
            }
        },
    },
}

```

```

action: function(args) {
    // Get the instance object of the selected row from context
    //
    // -- all currently loaded state is stored in 'context' as objects,
    //     such as the selected list view row,
    //     the selected section, and active user
    //
    // -- for list view actions, the object's key will be the same as
    //     listView.id, specified above;
    //     always make sure you specify an 'id' for the listView,
    //     or else it will be 'undefined!'
    var instance = args.context.testPluginInstances[0];

    plugin.ui.apiCall('rebootVirtualMachine', {
        // These will be appended to the API request
        //
        // i.e., rebootVirtualMachine&id=...
        data: {
            id: instance.id
        },
        success: function(json) {
            args.response.success({
                // This is an async job, so success here only indicates
                // that the job was initiated.
                //
                // To pass the job ID to the notification UI
                // (for checking to see when action is completed),
                // '_custom: { jobId: ... }' needs to always be passed on_
↪ success,

                // in the same format as below
                _custom: { jobId: json.rebootvirtualmachineresponse.jobid }
            });
        },

        error: function(errorMessage) {
            args.response.error(errorMessage); // Cancel action, show_
↪ error message returned
        }
    });

    // Because rebootVirtualMachine is an async job, we need to add
    // a poll function, which will periodically check
    // the management server to see if the job is ready
    // (via pollAsyncJobResult API call)
    //
    // The plugin API provides a helper function, 'plugin.ui.pollAsyncJob
↪ ',

    // which will work for most jobs
    // in CloudStack
    notification: {
        poll: plugin.ui.pollAsyncJob
    }
},

dataProvider: function(args) {

```

```
...
...
...
```

10. Add the thumbnail icon.

Create an icon file ; it should be square, about 50x50 pixels, and named *icon.png*. Copy it into the same directory with your plugin code : *cloudstack/ui/plugins/csMyFirstPlugin/icon.png*.

11. Add the stylesheet.

Create a CSS file, with the same name as your *.js* file. Copy it into the same directory with your plugin code : *cloudstack/ui/plugins/csMyFirstPlugin/csMyFirstPlugin.css*.

Allocators

CloudStack enables administrators to write custom allocators that will choose the Host to place a new guest and the storage host from which to allocate guest virtual disk images.

These are following categories of allocators currently supported :

- HostAllocators - Allows you to create custom rules to determine which physical host to allocate the guest virtual machines on.
- StoragePoolAllocators - Allows you to create custom rules to determine which storage pool to allocate the guest virtual machines on.

Implementing a custom HostAllocator

HostAllocators are written by extending `com.cloud.agent.manager allocator.HostAllocator` interface.

HostAllocator Interface

The interface defines the following two methods.

```
/**
 * Checks if the VM can be upgraded to the specified ServiceOffering
 * @param UserVm vm
 * @param ServiceOffering offering
 * @return boolean true if the VM can be upgraded
 */

public boolean isVirtualMachineUpgradable(final UserVm vm, final ServiceOffering
    offering);

/**
 * Determines which physical hosts are suitable to allocate the guest virtual
    machines on
 *
 * @param VirtualMachineProfile vmProfile
 * @param DeploymentPlan plan
 * @param Type type
 * @param ExcludeList avoid
 * @param int returnUpTo
 * @return List<Host> List of hosts that are suitable for VM allocation
 */

public List<Host> allocateTo( VirtualMachineProfile<?extendsVirtualMachine> vmProfile,
    DeploymentPlan plan, Type type, ExcludeList avoid, int returnUpTo);
```

A custom HostAllocator can be written by implementing the 'allocateTo' method

Input Parameters for the method 'HostAllocator : : allocateTo'

com.cloud.vm.VirtualMachineProfile vmProfile

VirtualMachineProfile describes one virtual machine. This allows the adapters like Allocators to process the information in the virtual machine and make determinations on what the virtual machine profile should look like before it is actually started on the hypervisor.

HostAllocators can make use of the following information present in the VirtualMachineProfile :

- The ServiceOffering that specifies configuration like requested CPU speed, RAM etc necessary for the guest VM.
- The VirtualMachineTemplate, the template to be used to start the VM.

com.cloud.deploy.DeploymentPlan plan

DeploymentPlan should specify :

- dataCenterId : The data center the VM should deploy in
- podId : The pod the Vm should deploy in ; null if no preference
- clusterId : The cluster the VM should deploy in ; null if no preference
- poolId : The storage pool the VM should be created in ; null if no preference

com.cloud.host.Host.Type type

Type of the Host needed for this guest VM. Currently com.cloud.host.Host.Type interface defines the following Host types :

- Storage
- Routing
- SecondaryStorage
- ConsoleProxy
- ExternalFirewall
- ExternalLoadBalancer

com.cloud.deploy.DeploymentPlanner.ExcludeList avoid

The ExcludeList specifies what datacenters, pods, clusters, hosts, storagePools should not be considered for allocating this guest VM. HostAllocators should avoid the hosts that are mentioned in ExcludeList.hostIds.

- Set Long dcIds ;
- Set Long podIds ;
- Set Long clusterIds ;
- Set Long hostIds ;
- Set Long poolIds ;

int returnUpTo

This specifies return up to that many available hosts for this guest VM.

To get all possible hosts, set this value to -1.

Reference HostAllocator implementation

Refer com.cloud.agent.manager allocator.impl.FirstFitAllocator that implements the HostAllocator interface. This allocator checks available hosts in the specified datacenter, Pod, Cluster and considering the given ServiceOffering requirements.

If returnUpTo = 1, this allocator would return the first Host that fits the requirements of the guest VM.

Loading a custom HostAllocator

1. Write a custom HostAllocator class, implementing the interface described above.
2. Package the code into a JAR file and make the JAR available in the classpath of the Management Server/tomcat.
3. Modify the components.xml and components-premium.xml files found in /client/ tomcatconf as follows.
4. Search for 'HostAllocator' in these files.

```
<adapters key="com.cloud.agent.manager allocator.HostAllocator">
  <adapter name="FirstFit" class="com.cloud.agent.manager allocator.impl.
↪FirstFitAllocator"/>
</adapters>
```

5. Replace the FirstFitAllocator with your class name. Optionally, you can change the name of the adapter as well.
6. Restart the Management Server.

Implementing a custom StoragePoolAllocator

StoragePoolAllocators are written by extending com.cloud.storage.allocator. StoragePoolAllocator interface.

StoragePoolAllocator Interface

A custom StoragePoolAllocator can be written by implementing the 'allocateTo' method.

```
/**
 * Determines which storage pools are suitable for the guest virtual machine
 * @param DiskProfile dskCh
 * @param VirtualMachineProfile vmProfile
 * @param DeploymentPlan plan
 * @param ExcludeList avoid
 * @param int returnUpTo
 * @return List<StoragePool> List of storage pools that are suitable for the VM
 */

public List<StoragePool> allocateToPool(DiskProfile dskCh, VirtualMachineProfile<?_
↪extends VirtualMachine> vm, DeploymentPlan plan, ExcludeList avoid, int returnUpTo);
```

This interface also contains some other methods to support some legacy code. However your custom allocator can extend the existing com.cloud.storage.allocator. AbstractStoragePoolAllocator. This class provides default implementation for all the other interface methods.

Input Parameters for the method 'StoragePoolAllocator : allocateTo'

com.cloud.vm.DiskProfile dskCh

DiskCharacteristics describes a disk and what functionality is required from it. It specifies the storage pool tags if any to be used while searching for a storage pool.

com.cloud.vm.VirtualMachineProfile vmProfile

VirtualMachineProfile describes one virtual machine. This allows the adapters like Allocators to process the information in the virtual machine and make determinations on what the virtual machine profile should look like before it is actually started on the hypervisor.

StoragePoolAllocators can make use of the following information present in the VirtualMachineProfile :

- The VirtualMachine instance that specifies properties of the guest VM.
- The VirtualMachineTemplate, the template to be used to start the VM.

com.cloud.deploy.DeploymentPlan plan

DeploymentPlan should specify :

- dataCenterId : The data center the VM should deploy in
- podId : The pod the VM should deploy in ; null if no preference
- clusterId : The cluster the VM should deploy in ; null if no preference
- poolId : The storage pool the VM should be created in ; null if no preference

com.cloud.deploy.DeploymentPlanner.ExcludeList avoid

The ExcludeList specifies what datacenters, pods, clusters, hosts, storagePools should not be considered for allocating this guest VM. StoragePoolAllocators should avoid the pools that are mentioned in ExcludeList.poolIds

- Set Long dcIds ;
- Set Long podIds ;
- Set Long clusterIds ;
- Set Long hostIds ;
- Set Long poolIds ;

int returnUpTo

This specifies return up to that many available pools for this guest VM

To get all possible pools, set this value to -1

Reference StoragePoolAllocator implementation

Refer `com.cloud.storage allocator.FirstFitStoragePoolAllocator` that implements the `StoragePoolAllocator` interface. This allocator checks available pools in the specified datacenter, Pod, Cluster and considering the given `DiskProfile` characteristics.

If `returnUpTo = 1`, this allocator would return the first Storage Pool that fits the requirements of the guest VM.

Loading a custom StoragePoolAllocator

1. Write a custom `StoragePoolAllocator` class, implementing the interface described above.
2. Package the code into a JAR file and make the JAR available in the classpath of the Management Server/tomcat.
3. Modify the `components.xml` and `components-premium.xml` files found in `/client/ tomcatconf` as follows.
4. Search for 'StoragePoolAllocator' in these files.

```
<adapters key="com.cloud.storage.allocator.StoragePoolAllocator">
  <adapter name="Storage" class="com.cloud.storage.allocator.
↪FirstFitStoragePoolAllocator"/>
</adapters>
```

5. Replace the `FirstFitStoragePoolAllocator` with your class name. Optionally, you can change the name of the adapter as well.
6. Restart the Management Server.

Deploying CloudStack with Ansible

In this article, [Paul Angus](#) Cloud Architect at ShapeBlue takes a look at using Ansible to Deploy an Apache CloudStack cloud.

What is Ansible

Ansible is a deployment and configuration management tool similar in intent to Chef and Puppet. It allows (usually) DevOps teams to orchestrate the deployment and configuration of their environments without having to re-write custom scripts to make changes.

Like Chef and Puppet, Ansible is designed to be idempotent. This means that you determine the state you want a host to be in and Ansible will decide if it needs to act in order to achieve that state.

There's already Chef and Puppet, so what's the fuss about Ansible ?

Let's take it as a given that configuration management makes life much easier (and is quite cool), Ansible only needs an SSH connection to the hosts that you're going to manage to get started. While Ansible requires Python 2.4 or greater on the host you're going to manage in order to leverage the vast majority of its functionality, it is able to connect to hosts which don't have Python installed in order to then install Python, so it's not really a problem. This greatly simplifies the deployment procedure for hosts, avoiding the need to pre-install agents onto the clients before the configuration management can take over.

Ansible will allow you to connect as any user to a managed host (with that user's privileges) or by using public/private keys – allowing fully automated management.

There also doesn't need to be a central server to run everything, as long as your playbooks and inventories are in-sync you can create as many Ansible servers as you need (generally a bit of Git pushing and pulling will do the trick).

Finally – its structure and language is pretty simple and clean. I've found it a bit tricky to get the syntax correct for variables in some circumstances, but otherwise I've found it one of the easier tools to get my head around.

So let's see something

For this example we're going to create an Ansible server which will then deploy a CloudStack server. Both of these servers will be CentOS 6.4 virtual machines.

Installing Ansible

Installing Ansible is blessedly easy. We generally prefer to use CentOS so to install Ansible you run the following commands on the Ansible server.

```
# rpm -ivh http://www.mirrorservice.org/sites/dl.fedoraproject.org/pub/epel/6/i386/  
→epel-release-6-8.noarch.rpm  
# yum install -y ansible
```

And that's it.

(There is a commercial version which has more features such as callback to request configurations and a RESTful API and also support. The installation of this is different)

By default Ansible uses /etc/ansible to store your playbooks, I tend to move it, but there's no real problem with using the default location. Create yourself a little directory structure to get started with. The documentation recommends something like this :

Playbooks

Ansible uses playbooks to specify the state in which you wish the target host to be in to be able to accomplish its role. Ansible playbooks are written in YAML format.

Modules

To get Ansible to do things you specify the hosts a playbook will act upon and then call modules and supply arguments which determine what Ansible will do to those hosts.

To keep things simple, this example is a cut-down version of a full deployment. This example creates a single management server with a local MySQL server and assumes you have your secondary storage already provisioned somewhere. For this example I'm also not going to include securing the MySQL server, configuring NTP or using Ansible to configure the networking on the hosts either. Although normally we'd use Ansible to do exactly that.

The pre-requisites to this CloudStack build are :

- A CentOS 6.4 host to install CloudStack on
- An IP address already assigned on the ACS management host
- The ACS management host should have a resolvable FQDN (either through DNS or the host file on the ACS management host)
- Internet connectivity on the ACS management host

Planning

The first step I use is to list all of the tasks I think I'll need and group them or split them into logical blocks. So for this deployment of CloudStack I'd start with :

- Configure selinux
- (libselinux-python required for Ansible to work with selinux enabled hosts)
- Install and configure MySQL
- (Python MySQL-DB required for Ansible MySQL module)
- Install cloud-client
- Seed secondary storage

Ansible is built around the idea of hosts having roles, so generally you would group or manage your hosts by their roles. So now to create some roles for these tasks

I've created :

- cloudstack-manager
- mysql

First up we need to tell Ansible where to find our CloudStack management host. In the root Ansible directory there is a file called 'hosts' (/etc/Ansible/hosts) add a section like this :

```
[acs-manager]
xxx.xxx.xxx.xxx
```

where xxx.xxx.xxx.xxx is the ip address of your ACS management host.

MySQL

So let's start with the MySQL server. We'll need to create a task within the mysql role directory called main.yml. The 'task' in this case to have MySQL running and configured on the target host. The contents of the file will look like this :

```
-name: Ensure mysql server is installed

yum: name=mysql-server state=present

- name: Ensure mysql python is installed

yum: name=MySQL-python state=present
```



```

- name: Ensure selinux python bindings are installed
yum: name=libselinux-python state=present

- name: Ensure cloudstack specific my.cnf lines are present
lineinfile: dest=/etc/my.cnf regexp='$item' insertafter="symbolic-links=0 line='$item'
with\_items:

- skip-name-resolve

- default-time-zone='+00:00'

- innodb\_rollback\_on\_timeout=1

- innodb\_lock\_wait\_timeout=600

- max\_connections=350

- log-bin=mysql-bin

- binlog-format = 'ROW'

- name: Ensure MySQL service is started
service: name=mysql state=started

- name: Ensure MySQL service is enabled at boot
service: name=mysql enabled=yes

- name: Ensure root password is set
mysql\_user: user=root password=$mysql\_root\_password host=localhost
ignore\_errors: true

- name: Ensure root has sufficient privileges
mysql\_user: login\_user=root login\_password=$mysql\_root\_password user=root host=%_
password=$mysql\_root\_password priv=*.\\*:GRANT,ALL state=present

```

This needs to be saved as `/etc/ansible/roles/mysql/tasks/main.yml`

As explained earlier, this playbook in fact describes the state of the host rather than setting out commands to be run. For instance, we specify certain lines which must be in the my.cnf file and allow Ansible to decide whether or not it needs to add them.

Most of the modules are self-explanatory once you see them, but to run through them briefly ;

The 'yum' module is used to specify which packages are required, the 'service' module controls the running of services, while the 'mysql_user' module controls mysql user configuration. The 'lineinfile' module controls the contents in a file.

We have a couple of variables which need declaring. You could do that within this playbook or its ‘parent’ playbook, or as a higher level variable. I’m going to declare them in a higher level playbook. More on this later.

That’s enough to provision a MySQL server. Now for the management server.

CloudStack Management server service

For the management server role we create a main.yml task like this :

```
- name: Ensure selinux python bindings are installed
  yum: name=libselinux-python state=present

- name: Ensure the Apache Cloudstack Repo file exists as per template
  template: src=cloudstack.repo.j2 dest=/etc/yum.repos.d/cloudstack.repo

- name: Ensure selinux is in permissive mode
  command: setenforce permissive

- name: Ensure selinux is set permanently
  selinux: policy=targeted state=permissive

- name: Ensure CloudStack packages are installed
  yum: name=cloud-client state=present

- name: Ensure vhdutil is in correct location
  get_url: url=http://download.cloud.com.s3.amazonaws.com/tools/vhd-util dest=/usr/
  ↪share/cloudstack-common/scripts/vm/hypervisor/xenserver/vhd-util mode=0755
```

Save this as */etc/ansible/roles/cloudstack-management/tasks/main.yml*

Now we have some new elements to deal with. The Ansible template module uses Jinja2 based templating. As we’re doing a simplified example here, the Jinja template for the cloudstack.repo won’t have any variables in it, so it would simply look like this :

```
[cloudstack]
name=cloudstack
baseurl=http://cloudstack.appt-get.eu/rhel/4.2/
enabled=1
gpgcheck=0
```

This is saved in */etc/ansible/roles/cloudstack-manager/templates/cloudstack.repo.j2*

That gives us the packages installed, we need to set up the database. To do this I’ve created a separate task called *setupdb.yml*

```
- name: cloudstack-setup-databases
  command: /usr/bin/cloudstack-setup-databases cloud:{{mysql\_cloud\_password }}
  ↪@localhost --deploy-as=root:{{mysql\_root\_password }}
```

```
- name: Setup CloudStack manager
command: /usr/bin/cloudstack-setup-management
```

Save this as `/etc/ansible/roles/cloudstack-management/tasks/setupdb.yml`

As there isn't (as yet) a CloudStack module, Ansible doesn't inherently know whether or not the databases have already been provisioned, therefore this step is not currently idempotent and will overwrite any previously provisioned databases.

There are some more variables here for us to declare later.

System VM Templates :

Finally we would want to seed the system VM templates into the secondary storage. The playbook for this would look as follows :

```
- name: Ensure secondary storage mount exists
  file: path={{ tmp\_nfs\_path }} state=directory

- name: Ensure NFS storage is mounted
  mount: name={{ tmp\_nfs\_path }} src={{ sec\_nfs\_ip }}:{{ sec\_nfs\_path }}
  ↪fstype=nfs state=mounted opts=nolock

- name: Seed secondary storage
  command:
  /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-templt -m {{
  ↪tmp\_nfs\_path }} -u http://download.cloud.com/templates/4.2/systemvmtemplate-2013-
  ↪06-12-master-kvm.qcow2.bz2 -h kvm -F

  command:
  /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-templt -m {{
  ↪tmp\_nfs\_path }} -u http://download.cloud.com/templates/4.2/systemvmtemplate-2013-
  ↪07-12-master-xen.vhd.bz2 -h xenserver -F

  command:
  /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-templt -m {{
  ↪tmp\_nfs\_path }} -u http://download.cloud.com/templates/4.2/systemvmtemplate-4.2-
  ↪vh7.ov -h vmware -F
```

Save this as `/etc/ansible/roles/cloudstack-manager/tasks/seedstorage.yml`

Again, there isn't a CloudStack module so Ansible will always run this even if the secondary storage already has the templates in it.

Bringing it all together

Ansible can use playbooks which run other playbooks, this allows us to group these playbooks together and declare variables across all of the individual playbooks. So in the Ansible playbook directory create a file called `deploy-cloudstack.yml`, which would look like this :

```
-hosts: acs-manager

vars:
```

```
mysql\_root\_password: Cl0ud5tack
mysql\_cloud\_password: Cl0ud5tack
tmp\_nfs\_path: /mnt/secondary
sec\_nfs\_ip: IP\_OF\_YOUR\_SECONDARY\_STORAGE
sec\_nfs\_path: PATH\_TO\_YOUR\_SECONDARY\_STORAGE\_MOUNT

roles:

- mysql
- cloudstack-manager

tasks:

- include: /etc/ansible/roles/cloudstack-manager/tasks/setupdb.yml
- include: /etc/ansible/roles/cloudstack-manager/tasks/seedstorage.yml
```

Save this as `/etc/ansible/deploy-cloudstack.yml` inserting the IP address and path for your secondary storage and changing the passwords if you wish to.

To run this go to the Ansible directory (`cd /etc/ansible`) and run :

```
# ansible-playbook deploy-cloudstack.yml -k
```

‘-k’ tells Ansible to ask you for the root password to connect to the remote host.

Now log in to the CloudStack UI on the new management server.

How is this example different from a production deployment ?

In a production deployment, the Ansible playbooks would configure multiple management servers connected to master/slave replicating MySQL databases along with any other infrastructure components required and deploy and configure the hypervisor hosts. We would also have a dedicated file describing the hosts in the environment and a dedicated file containing variables which describe the environment.

The advantage of using a configuration management tool such as Ansible is that we can specify components like the MySQL database VIP once and use it multiple times when configuring the MySQL server itself and other components which need to use that information.

Acknowledgements

Thanks to Shanker Balan for introducing me to Ansible and a load of handy hints along the way.