Intel® Cloud Builder Guide to Cloud Design and Deployment on Intel® Platforms

Ubuntu Enterprise Cloud

White Paper

Intel® Cloud Builder Guide

Intel® Xeon® Processor

Ubuntu Enteprise Cloud

Canonical Ltd.

Eucalyptus Systems, Inc.

CANONICAL

Eucalyptus Systems

Audience and Purpose of This Paper

For Cloud Service Providers, Hosters and Enterprise IT who are looking to build their own cloud infrastructure, the decision to use a cloud for the delivery of IT services is best done by starting with the knowledge and experience gained from previous work. This white paper gathers into one place a complete example of running a Canonical Ubuntu Enterprise Cloud on Intel®-based servers and is complete with detailed scripts and screen shots. Using the contents in this paper should significantly reduce the learning curve for building and operating your first cloud computing instance.

Since the creation and operation of a cloud requires integration and customization to existing IT infrastructure and business requirements, it is not expected that this paper can be used as-is. For example, adapting to existing network and identify management requirements are out of scope for this paper. Therefore, it is expected that the user of this paper will make significant adjustments to the design to meet specific customer requirements. This paper is assumed to be a starting point for that journey.

Table of Contents

Executive Summary

The operation of IT using the ideas and principles of the cloud are of great interest because of the benefits of agility and cost savings. For those workloads that need to remain under close control of IT, using an IaaS structure is often the best choice. When the IaaS (Infrastructure as a Service) is hosted internal to the IT (a Private Cloud) the next question is how to build this cloud. Given the many choices, trade-offs, and decisions, it is clear that being able to start from a known and understood point is key. This white paper summarizes a body of work on building a cloud leveraging the Intel® Xeon® processor 5500 series (codenamed Nehalem) and the Ubuntu* Enterprise Cloud (UEC) software from Canonical powered by Eucalyptus Systems Inc. The three companies worked together to construct the cloud described in this white paper and offer this as a starting point for building and operating a cloud.

Our cloud implementation is a straightforward, albeit small, cloud: 12 servers and 1.4 TB of storage exposed as a compute service and a storage service. The interface is similar to that which is exposed by Amazon Web Services* and as implemented by Eucalyptus. We placed an emphasis on being able to host virtual machines (VMs) for a number of logical customers, therefore multi-tenancy, or the isolation of compute, storage and network was important. Other design requirements included self-contained storage and the ability to delegate management of the logical customer's compute and storage to the customer, aka a self service portal.

Our design, as shown in Figure 1, is a simple Eucalyptus setup with two clusters of compute and a single storage pool. The specific arrangement shown in the figure below allows for the creation of 128 virtual machines with associated storage (VMs and data). This design supports the implementation of multi-tenancy through the use of KVM (for compute isolation) and virtual dynamic VLAN (to isolate network traffic).

With this configuration, we were able to start and stop VMs using command-line tools and connect to the virtual machine via SSH (Secure Shell). The use of access keys allowed a "customer" to create, start, stop, terminate an instance independent of other "customers" in the same cloud. All VM images were stored in the Walrus Storage Service. The actual storage behind the Walrus was implemented as iSCSI volumes hosted on the Storage Server. Network traffic was managed by the Cluster Controller which provides the routing function to allow hosted VMs access to/from the Internet and access to the Walrus storage.

The Node Controllers were implemented using Intel Xeon processor 5500 series-based servers. The Intel Xeon processor 5500 series-based servers are ideal for highly dense cloud architecture applications because of the optimized power foot print and Intel® Intelligent Power Node Manager, which provides for node-level power management policy and enforcement. Intel® Dynamic Data Center Manager was also used to simplify power management across the entire collection of nodes for the purpose of power capping or to optimize power usage.

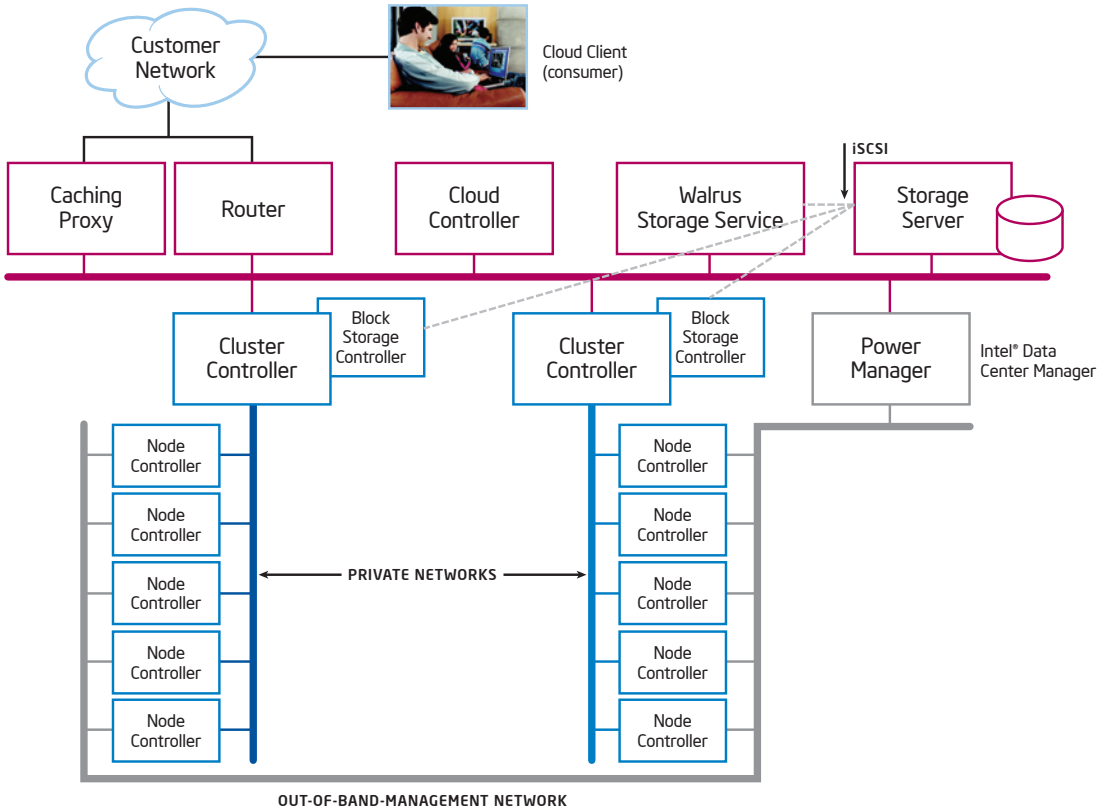


Figure 1. Reference Cloud Design using Eucalyptus

Introduction

The operation of a cloud is often driven by the need to lower operational cost and/or to improve the ability of IT to respond to sudden changes in demand. To achieve that goal, the applications in the cloud either need to change to directly leverage the capabilities in the cloud using distributed computing tools such as Apache Hadoop* or the applications need to be virtualized so that the cloud infrastructure can use the flexibility that virtualization provides to enable optimal placement and recovery in the event of failures.

To achieve lowest operational costs for the infrastructure, a homogeneous pool of compute and storage with uniform connectivity is the simplest to manage, easiest to troubleshoot, and is the easiest for adding/removing capacity. This is not to say that a cloud contains only one kind of server or storage elements. But, instead, we strive to create pools of uniform compute, storage, and networking to allow new pools to be added over time, say as new generations of server technology becomes available. This design approach allows for the automation of the placement of workloads, the connection to external networks and storage, and for the delegation of the management of the workloads in the cloud (because bare-metal access is not required, OS-based methods for access can be used uniformly).

Hadoop* is a Java software framework that supports data-intensive distributed applications under a free license. Hadoop enables applications to work with thousands of nodes and petabytes of data.

Lastly, we need to be able to make the assumption that things will fail: hardware, software, network, storage, etc. Therefore, we need either the application or the cloud management software to be able to perform the necessary recovery actions on the occurrence of failure. This requires that the state for each compute element be maintained in shared storage or that the application accept the responsibility for retrying the actions on a failing device. For the typical non-cloud-aware enterprise workload this means that the application is wrapped in a VM and that the VM and all data are stored on shared storage. This way, if a server fails, the cloud management software can simply restart the VM on another server.

We used these principles in design and implemented the UEC reference architecture in the Intel® Cloud Testbed (see Figure 2). This diagram approximates the physical construction of the cloud.

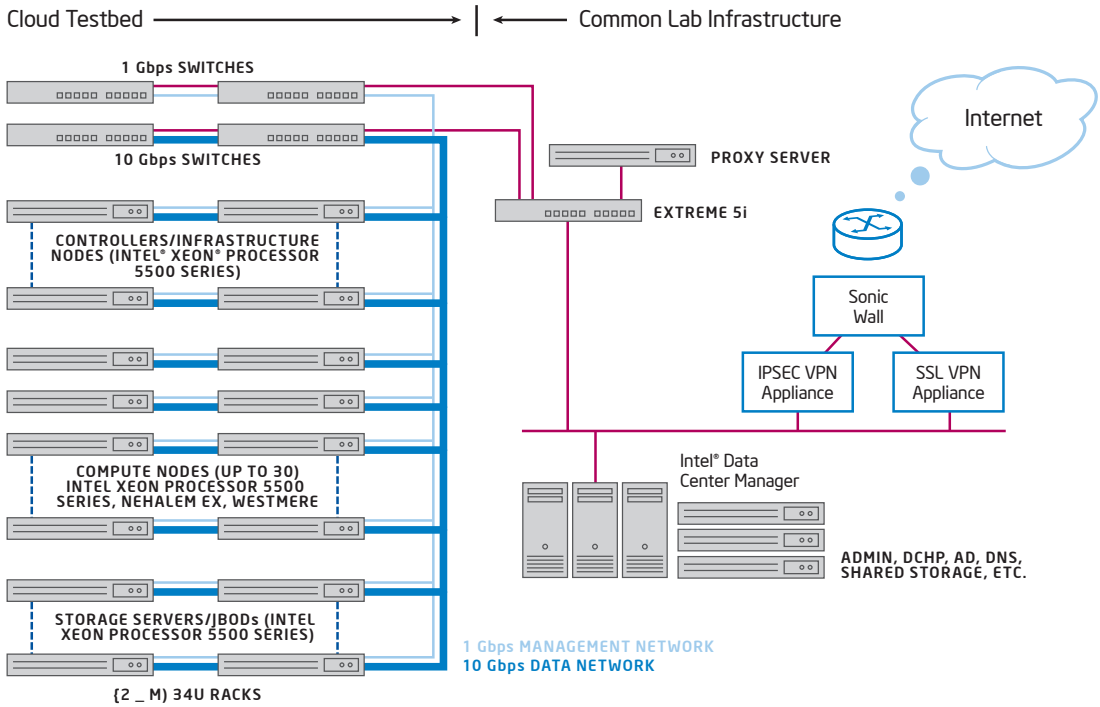


Figure 2. Physical Cloud Infrastructure

A set of 12 Intel Xeon processor 5500 series-based servers are connected using a 1G network to a 1G "top of rack" switch. The switch was configured to achieve the logical architecture shown in Figure 3. In this configuration, there are no "special" servers and no special connections – all servers are configured the same. This uniformity allows for simple replacement, or reassignment of workloads.

Some of the design considerations used in this design are:

- The use of the Cluster Controllers to manage the network traffic is a key component of the multi-tenant design. While the KVM hypervisor provides isolation for the workloads on the servers, the network traffic isolation between virtual machines is implemented by network packet tagging and routing by the Cluster Controllers.
- The use of standard Intel® server building blocks means that any server in the rack can be used for any purpose (Cluster Controller, Walrus Storage Service, etc.).
- We chose to use a flat layer 2 network to reduce the implementation cost and to retain the flexibility to assign any workload to any server. This approach is flexible at the cluster level but may create bottlenecks as this design is aggregated at the data center level. Therefore, the Cloud Controller will need to assign workloads with the goal of keeping related workloads together and close to their required data.
- We chose to implement the Walrus controller using local DAS (Direct Attach Storage). This was merely a convenient and low-cost way to get a large quantity of storage capacity accessible to the clusters. Other implementations could connect the Walrus server to a SAN (Storage Area Network) or NAS (Network Attached Storage) device and not require local DAS storage.

This implementation provided all the features (e.g., Provisioning, Monitoring, virtual machine management, etc.) to deliver the required IaaS functionality. The following sections provide the details on the software implementation, the hardware testbed, the test cases that we ran, and some of the things to consider on the design a cloud.

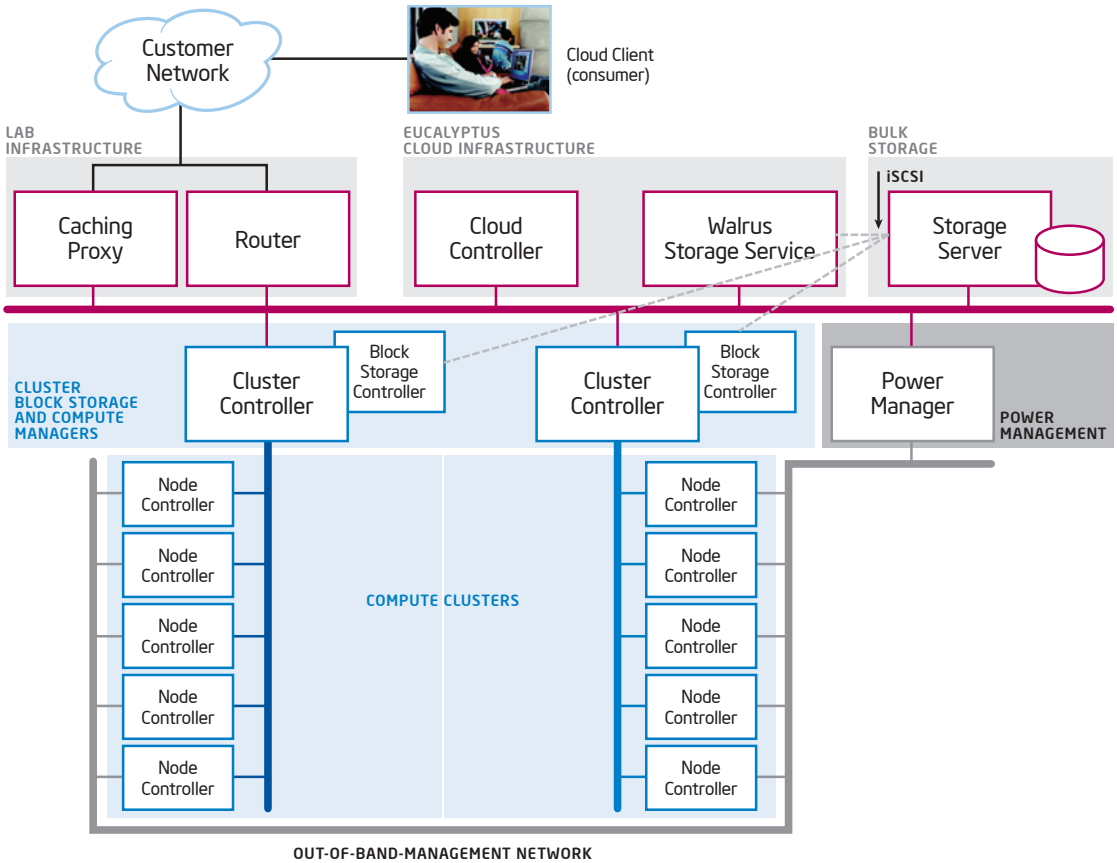


Figure 3. Eucalyptus Logical Architecture

UEC Implementation Overview

Ubuntu Enterprise Cloud is an open-source software stack that complements Eucalyptus, allowing for the implementation of a private cloud infrastructure using a combination of Canonical and Eucalyptus Systems technologies. The Ubuntu choice of Eucalyptus was made after a careful study of the cloud market, identifying the EC2, EBS and S3 API access to the Amazon Web Services* (AWS) as a widely used industry interface, and selecting the best open-source component to reproduce it.

Eucalyptus Systems provides and supports an open-source cloud platform that obeys the same API that AWS uses, but in a way that automatically translates Amazons AMI image format to the image format required by the local hypervisor. Thus, a Eucalyptus cloud is API compatible with AWS regardless of the underlying virtualization technology that is installed. This is, in itself, a first validation of the independence that open-source provides in the cloud space. With Ubuntu 9.04 Server Edition,* Canonical delivered its first integration of Eucalyptus with its own installation and image repository enhancements for Ubuntu and known as the UEC. This technology preview allowed deployment of an internal cloud that obeys to the same API that AWS uses and with an image format that would allow us to ensure compatibility with AWS. Following up this initial release, Canonical, the commercial sponsor of Ubuntu, announced a series of services from consulting to support and training, specific to the cloud to help the organizations to deploy Ubuntu Enterprise Cloud.

Ubuntu 9.10 Server Edition was the first release to include Ubuntu Enterprise Cloud and Eucalyptus as a fully supported and maintained option, with a very simple mechanism to deploy it from the installer.

In order for the Ubuntu Enterprise Cloud to make best use of the hardware that was offered by the testbed and to support the planned test cases, the installation of Eucalyptus was designed so that:

- Most components of the architecture could be installed on separate hardware.
- Consideration would be given on maximum bandwidth usage and potential bottlenecks.
- Future potential enhancement of the platform (increasing number of cores per server) can help achieve higher scalability.

The design comprises the following elements:

- The general lab infrastructure including the Caching Proxy, a router, and Power Management (Intel® Data Center Manager). A Caching Proxy (Squid) was used to accelerate the update process for each of the Ubuntu Server instances.
- The Cloud Controller provides the primary interface point for interacting with the cloud. Commands to create or terminate virtual machines are initiated through the API interface at the Cloud Controller.
- The Walrus Storage Service exposes the object store. The object store is used to hold the virtual machine images prior to instantiation and to hold user data.
- The Storage Server hosts the actual bulk storage (a 1.4 TB JBOD in this case). Storage is exposed to the Block Storage Controllers and the Walrus Controller as a set of iSCSI volumes.
- The Cluster Controllers manage a collection of Node Controllers and provide the traffic isolation.
- The Block Storage Controllers (SCs) manage dynamic block devices (e.g., EBS) that VMs can use for persistent storage.
- And, the Node Controllers (NCs) which are the servers in the pools that comprise the compute elements of the cloud.

Together, these elements implement the cloud for the purposes of this white paper.

Cluster Design Overview

In this design, we have chosen to use a Class C address space for each cluster for simplicity. Therefore, each cluster is limited to 254 machine instances (one IP address per VM instance). Eucalyptus allows its administrator to configure the degree of multi-tenancy (i.e., the number of VMs hosted per machine) that the cloud should implement. In our Eucalyptus installation, we have chosen to allocate no more than 16 "small" machine instances per server. It is likely that this configuration will result in underutilization of the server, but it conforms to the published AWS API specification and simplifies our design. The number of VM instances allowed per server should be a function of the memory in the system (since memory is partitioned among instances), the performance of the system, and the load generated by the VM instances.

To allow for multi-tenancy network security, Eucalyptus supports a Managed/VLAN networking mode. This mode transforms the Cluster Controller into a router for all VMs running in the cluster (aka availability zone) that it controls. This allows the Cluster Controller to dynamically create tagged VLANs on its private network that it fully supervises. As a consequence of this, and also to ensure maximum bandwidth availability, each cluster must have its own dedicated physical switch or segment of a switch. Additional scaling of our cloud is easily accomplished by the addition of additional 10 nodes clusters on separate Cluster Controller.

Storage Design Overview

Ubuntu Enterprise Cloud uses two components for storage:

- Walrus, which is a functional equivalent of Amazon S3, and
- Block Storage Controller, which is a functional equivalent of Amazon Elastic Block Storage (EBS).

Our storage design used a single large store for all cloud storage. In a commercial implementation this would provide a convenient place for backup and replication. For our purposes, the Storage Server exposed a set of block devices via iSCSI which could be mounted as volumes in the Walrus or Block Storage Controllers.

Each of the Node Controllers has a local storage device (DAS). The local storage was only used to hold the VM image at run time and for caching VM instances. When a VM instance is terminated, the storage on the Node Controller local storage is released and is, therefore, not persistent. To maintain persistent data, either a volume from the Block Storage Manager must be used or the applica-tion must be designed to use the Walrus object store.

Images and Instances

Each user of the infrastructure prepares machine images and instantiates as many times as the infrastructure permits. Since the infrastructure is multi-tenant, multiple machine images from multiple users can be running on the same infrastructure within their own virtual environment with the hypervisor (KVM), the storage controllers, and the network providing the isolation. Once a user is approved to use the infrastructure, instances can be started or stopped at the request of the user without the need for the operator of the cloud to get involved.

This infrastructure uses the same API as Amazon EC2. From the point of view of the user, this API implements semantics which can be a bit surprising to the first-time users:

- Instances are not persistent across a server hardware reboot
- Machine images are the only persistent form of a machine
- Administrators define base architectures for machine image allocation (32 or 64 bits, RAM, etc.), and users decide which architecture they need
- Persistent storage is available through other mechanisms such as Walrus Storage Service or the Block Storage Controller

The Walrus Storage Service implements an object store. In an object store, an item is stored as a named object using a key. When the object is to be retrieved, the key is used to identify the object which is subsequently returned. By contrast, the Block Storage Controller presents mountable volumes to the VMs running in the cluster. The volumes presented in this way provide a place for persistent data to be stored for the VM instances.

The idea is that users prepare templates of VMs (which are called machine images) which can be instantiated as many times as needed. A machine image is a template while a machine instance is an actual instantiation of the template. When the instance is terminated, the state is lost unless previously saved to disk, exactly the same way a machine instance is lost if the server is unexpected switched off.

Users can instantiate a machine image, modify the running server configuration, and then save the result as a new machine image. This operation is called re-bundling. When an image is instantiated, it is possible to pass it a block of data, which the instance can then use during initialization. A common practice is to create a base OS image template (such as the freely available yet maintained Ubuntu Server Edition), an invoke a script that customizes the instance based on the data that was passed to it. Since the process of re-bundling can be time consuming, it is common practice to use the capability of Eucalyptus to pass scripts and data to an instance at startup to allow for run time customization of an instance and avoid the need to continually re-bundle.

It should also be noted that Eucalyptus automatically converts Amazon's AMI format to one that is compatible with KVM. This means that the same image can be moved from UEC's private cloud to Amazon's public one without any modifications, thus allowing hybrid cloud designs where instances from one cloud can also be started in Amazon's cloud.

Testbed Blueprint Overview

The cloud architecture highlighted in the following diagram is typical of a cloud data center design: highly efficient Intel Xeon processor 5500 series-based servers, flat layer 2 networks, consolidated storage, and highly virtualized. These attributes provide for the cost-effective operation and supports a highly automated infrastructure.

Hardware Description

We used Intel's latest innovation in processor technology, the Intel Xeon processor 5500 series code named Nehalem which provides a foundation for designing new cloud data centers to achieve greater performance while using less energy and space, and dramatically reducing operating costs.

The Intel Xeon processor 5500 series offers several features that help it make the best performing server in the industry. Some of these features include:

1. Intelligent performance that automatically varies the processor frequency to meet the business and application performance requirements.
2. Automated energy efficiency that scales energy usage to the workload to achieve optimal performance/watt and reduce operating costs.
3. Intel® Virtualization Technology◊ (Intel® VT) and Intel® VT FlexMigration offer best-in-class performance and manageability in virtualized environments to strengthen the infrastructure and reduce costs.
4. Intel® Hyper-Threading Technology,† with eight computation engines and 16 threads per 2-socket platform, helps multi-threaded software applications to execute threads in parallel within each processor core.

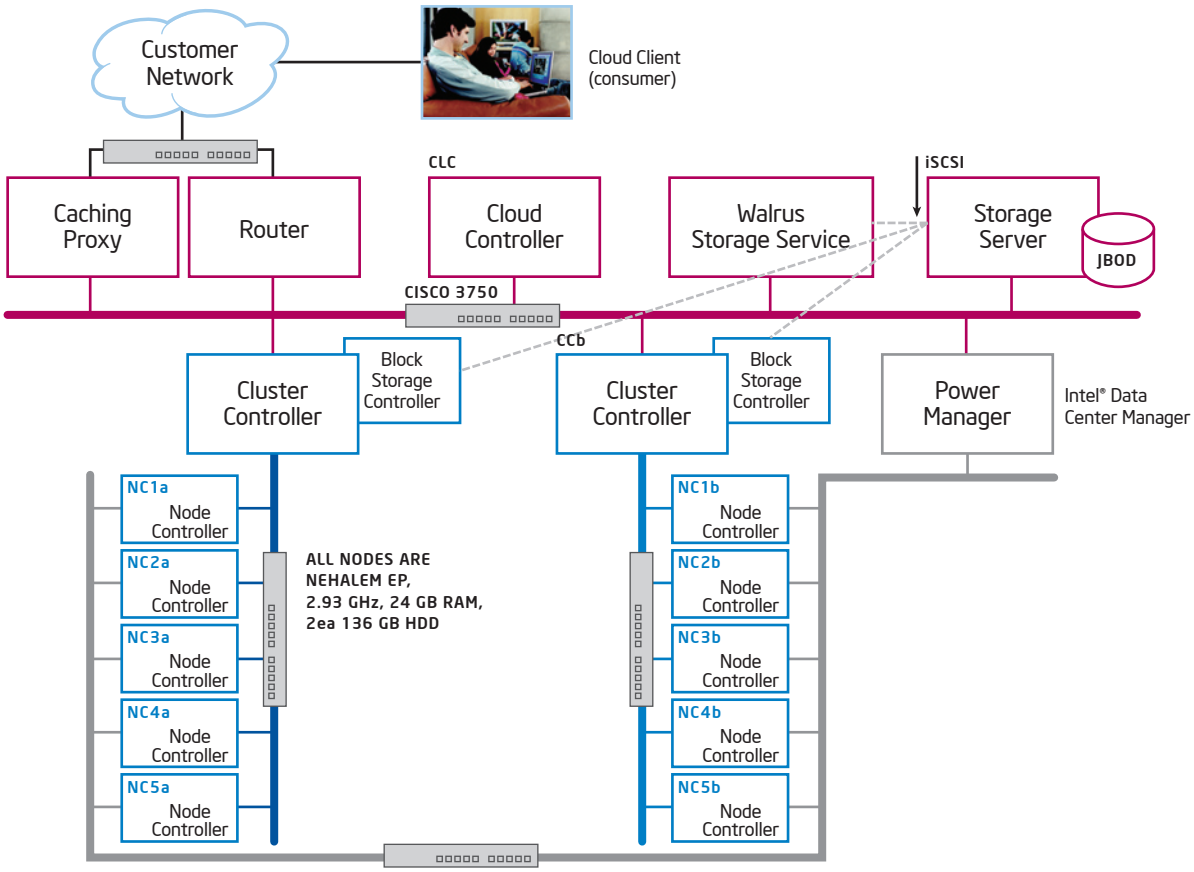


Figure 4. Testbed Physical Topology

Table 1. Cloud Testbed Systems Configuration

System	Processor Configuration	Other Info
Cloud Controller [CLC] Cluster Controller(CC) and Block Storage Controller [CCa and CCb]	Intel® Xeon® Processor X5570Δ	▪ **Form Factor:** 2U Rack Mount Server ▪ **Processor:** Intel® Xeon® processor 5500-based series 2.93 GHz; 2-way x 4 cores = 8 cores ▪ **Memory:** 24 GB RAM ▪ **Storage:** 300 GB HDD
Node Controller [NC1a-5a and NC1b-NC5b]	Intel Xeon Processor X5570	▪ **Form Factor:** 1U Rack Mount Server ▪ **Processor:** Intel Xeon processor 5500-based series 2.93 GHz; 2-way x 4 cores = 8 cores ▪ **Memory:** 24 GB RAM ▪ **Storage:** 136 GB HDD
Walrus Storage Service	Intel Xeon Processor X5570	▪ **Form Factor:** 2U Rack Mount Server ▪ **Processor:** Intel Xeon processor 5500-based series 2.93 GHz; 2-way x 4 cores = 8 cores ▪ **Memory:** 48 GB RAM ▪ **Storage:** 300 GB HDD
Storage Server	Intel Xeon Processor X5570	▪ **Form Factor:** 5U Tower ▪ **Processor:** Intel Xeon processor 5500-based series 2.93 GHz; 2-way x 4 cores = 8 cores ▪ **Memory:** 24 GB RAM ▪ **Storage:** 6x300 GB HDD (1.4 TB RAID)
Proxy Server	Intel® Xeon® Processor 5140Δ	▪ **Form Factor:** 1U Rack Mount Server ▪ **Processor:** Intel® Xeon® processor 5100-based series 2.33 GHz Xeon; 2-way ▪ **Memory:** 4 GB RAM ▪ **Storage:** 4x68 GB HDD

Table 2. Allocated IP Addresses

	IP range	Subnet mask	Broadcast
Cloud Network	192.168.16.0/20	255.255.240.0	192.168.31.255
Private VM network	10.0.0.0/8	255.0.0.0	10.255.255.255
ClusterController[CCa]	192.168.32.0/24	255.255.255.0	192.168.32.255
ClusterController[CCb]	192.168.33.0/24	255.255.255.0	192.168.33.255
CCa Public IP for instance	192.168.17.1/254	255.255.255.0	192.168.17.255
CCb Public IP for instance	192.168.18.1/254	255.255.255.0	192.168.18.255

Table 1 shows the configuration of the systems that were employed in building the cloud testbed.

System Design

Following the basic tenet of cloud design (keep it simple), we partitioned the cloud design into three functional blocks: cloud controller and infrastructure control, storage elements (Walrus Storage Service and Block Storage Controllers), and compute elements (Cluster Controllers and Node Controllers). We then allocated the server types listed above such that the servers with JBODs were used in the storage elements and all other servers were used as needed. Then, we partitioned the network into five networks:

- One network for the customer network, this is the means by which the customer accesses the cloud
- One network internal to the cloud, this is the network over which all command and control functions occur from the Cloud Controller to the Cluster Controllers
- Two networks, one for each of the clusters
- One network for "out of band" access to the servers, this network is used for power on/off control and to implement power management

For network configuration, we used three Cisco 3750G* (1G) switches. All switches were "out of the box" (i.e., no VPN configuration with all the 48 ports configured onto a single subnet) with a total of three subnets. We used 1G switches in this design. However, for scalability and bandwidth considerations, 10G is recommended. This is due to the fact that all data traffic is concentrated in the cloud network due to the high density of VMs.

The IP addresses allocated for the networks are shown in Table 2.

Our design reserves static IP addresses for specific nodes: the Cloud Controller and Cluster Controllers and the servers in the lab infrastructure. Node Controllers and VM instances used IP addresses obtained from their respective Cluster Controller.

In the lab infrastructure, we configured a server using Squid to act as a caching proxy server. This was done specifically to speed the process by which the servers were able to get up and running. Since each server is installed with a clean copy of Ubuntu Enterprise Server, each server will need to get updates and packages from the Internet. The Caching Proxy (Squid) server is configured to cache these packages to speed up the process.

Technical Review

This section provides a detailed overview of the actions performed to implement and operate the cloud. Many of these commands are unique to our particular design but should provide enough detail to understand what we did and also to help recreate the cloud design. For the details on the command and configuration files, please refer to the Canonical UEC and Eucalyptus documentation.

Server Setup and Configuration Information

Ubuntu Server Installation

Every server in the system had Ubuntu Server Edition installed and each installation was identical. Only the server name and the packages loaded on each server were varied. Therefore, the process below for installing Ubuntu is defined once here, and referred to in each of the sections that follow. For a more generic installation guide please see the Ubuntu Enterprise Cloud documentation.

We started each server with the default BIOS settings and selected to boot using a USB drive onto which the Ubuntu Server image (64 bit) is installed. The image can be downloaded from www.ubuntu.com/getubuntu/download-server and installed on the USB drive using the instructions from help.ubuntu.com/community/Installation/FromUSBStick

Notes:

- Be sure to configure the server name and network IP address, static or dynamic, appropriately for the server type as noted below.
- Be sure to note the account name and password for later use. Note that Ubuntu, by default, there is no root account password.
- Select automatic updates so that the latest patches will be obtained automatically.

```
:$ sudo apt-get update
:$ sudo apt-get dist-upgrade
:$ sudo ntpdate proxy.lab.clbp.com
:$ sudo apt-get install ntp openssh-server
:$ sudo vi/etc/ntp.conf
```

server proxy.lab.clbp.com

```
:$ sudo service ntp restart
```

Proxy/DNS/NTP Installation

Server Information:

- Server name: Proxy
  - IP Address: 192.168.16.11

> In Linux* (and UNIX* in general), there is a superuser-named root. The Windows equivalent of root is Administrator. The superuser can do anything and everything, and thus doing daily work as the superuser can be dangerous. You could type a command incorrectly and destroy the system. Ideally, you run as a user that has only the privileges needed for the task at hand. In some cases, this is necessarily root, but most of the time it is a regular user.
>
> By default, the root account password is locked in Ubuntu. This means that you cannot log in as root directly or use the sudo command to become the root user. However, since the root account physically exists it is still possible to run programs with root-level privileges. This is where sudo comes in – it allows authorized users to run certain programs as root without having to know the root password.

DNS-Bind Configuration:

We installed the package "Bind" in the DNS server by following the below command sequence.

```
:$sudo apt-get install bind9
```

We then added the zones and IP addresses for selected servers.

```
:$sudo vi /etc/bind/named.conf.local
Zone "lab.clbp.com" in
{ Type master; file
"/etc/bind/db.lab.clbp.com"; };
proxy 1h A 192.168.16.11
san   1h A  192.168.16.10
```

Finally, we restarted the DNS server.

```
:$sudo /etc/init.d/bind9 restart
```

Squid-Proxy Server Configuration:

Using Squid to cache Ubuntu packages allows us to avoid re-downloading the same package over the Internet. The Squid configuration that we used is based on the Ubuntu wiki configuration.

We installed the caching proxy server (Squid).

```
:$sudo apt-get install squid squid-common
```

You will need to modify the Squid settings to listen on port 80, to allow only certain users on the local network access to the Internet, to allow all HTTP access, and enable the cache.

```
:$sudo vi /etc/bind/squid.conf
```

We defined the port for listening for requests:

```
http _ port 80
```

Using Squid's access control, you may configure use of Internet services proxied by Squid to be available only to users with certain Internet Protocol (IP) addresses. For example, we will illustrate access by users of the 192.168.42.0/24 subnet only.

We added the following to the **bottom** of the ACL section of the /etc/squid/squid.conf file:

```
acl localnet src 192.168.0.0/16 # RFC1918 possible
internal network
```

Next, we added the following to the top of the http_access section of the /etc/squid/squid.conf file:

```
http _ access allow all
```

Edited the cache_dir [optional]

```
Cache _ dir ufs /var/spool/squid3 60000 16 256
Maximum _ object _ size 100000 KB
```

Modified the refresh pattern to suit Ubuntu packages

```
# refresh pattern for debs and udebs
refresh _ pattern deb$   129600 100% 129600
refresh _ pattern udeb$   129600 100% 129600
refresh _ pattern tar.gz$  129600 100% 129600

# handle meta-release and changelogs.ubuntu.com
special
refresh _ pattern changelogs.ubuntu.com/*  0  1% 1
```

And restarted the squid service to make the changes effective.

```
:$ sudo /etc/init.d/squid restart
```

Cloud Controller (CLC) Installation

Server Information:

- Server name: CLC
- IP Address: 192.168.16.2

Cloud Controller Installation:

We installed the Eucalyptus cloud controller software.

```
:$ sudo apt-get install eucalyptus-cloud euca2ools
openssh-server screen nmap
```

And entered postfix Internet mail server in the wizard and made sure process Eucalyptus-cloud was running.

```
:$ ps -edf |grep eucalyptus-cloud
```

Walrus Storage Service Installation

Server Information:

- Server name: Walrus
- IP Address: 192.168.16.3

Walrus Installation Steps:

We installed the Walrus software.

```
:$ sudo apt-get install eucalyptus-walrus
```

And made sure that Walrus services were running by using the following command.

```
:$ ps -ax | grep eucalyptus
```

Cluster Controller (CC) and Block Storage Controller Installation

Server Information:

- Server name: CCa and CCb
- IP Address: 192.168.16.4, 192.168.16.5

Cluster Controller Installation Steps:

We installed the Cluster Controller and the Storage Controller software.

```
:$ sudo apt-get install eucalyptus-cc
eucalyptus-sc
```

Next, we entered the name of the new cluster, and finally supplied the public IP range to use:

```
Cluster A: 192.168.17.1 - 192.168.17.254
Cluster B: 192.168.18.1 - 192.168.18.254
```

We then installed multi-cluster software dependencies

```
:$ apt-get install bridge-utils vtun
```

Cluster Controller Configuration

We edited /etc/network/interfaces to add eth1 as an interface since this machine will act as a router. In the installation we were developing, eth1 is connected to the private network. See the eucalyptus.conf files for reference in the appendix.

We configured the NIC dedicated to cloud tasks (private network) with a bridge. The bridge is needed in a multi-cluster setup.

We restarted the service//
:$ sudo /etc/init.d/networking restart

We edited eucalyptus-ipaddr.conf on CCa and CCb. We changed the ip addr of the SC and CCa private interface manually to allow for automatic discovery of the services using the Avahi protocol.

We edited eucalyptus.conf on CCa and CCb and added VNET_CLOUDIP="192.168.16.2"

We then restarted the service with a config reinitialization

```
:$ sudo restart eucalyptus-cc CLEAN=1
```

Component Registration:
The Cloud Controller must be aware of all components before operation can begin.

Here are the steps we used for setting password-less login, registration and distribution of credentials:

On the Cloud Controller, Walrus, and both Cluster Controller servers, we set up a password for user Eucalyptus.

```
:$ sudo passwd eucalyptus
```

Next, on the Cloud Controller, we set up ssh key-based password-less login for user Eucalyptus.

```
:$ sudo -u eucalyptus ssh-copy-id -i
~eucalyptus/.ssh/id _ rsa.pub eucalyptus@<WALIP>
:$ sudo -u eucalyptus ssh-copy-id -i
~eucalyptus/.ssh/id _ rsa.pub eucalyptus@<CCAIP>
```

We removed the password for user Eucalyptus on the Cloud Controller, Walrus, and Cluster Controller servers.

```
:$ sudo passwd -d eucalyptus
```

To register the components, we used the following on the Cloud Controller:

```
:$ sudo euca _ conf --no-rsync --register-walrus
<WALIP>
:$ sudo euca _ conf --no-rsync --register-cluster
<clusterAname> <CCAIP>
:$ sudo euca _ conf --no-rsync --register-sc <cluster-
Aname> <CCAIP>
```

We listed components just to be sure everything is registered.

```
:$ sudo euca _ conf --list-walruses
:$ sudo euca _ conf --list-clusters
:$ sudo euca _ conf --list-scs
```

On the Cloud Controller, we next obtained credentials to be used with the administration tools.

```
:$ mkdir -p ~/.euca
:$ chmod 700 ~/.euca
:$ cd ~/.euca
:$ sudo euca _ conf --get-credentials mycreds.zip
:$ unzip mycreds.zip
:$ cd –
```

Extracting and using your credentials
The next part of the installation involves setting up the EC2 API and AMI tools on the machine where they would be run. Generally, this will be on the developers machine.

We sourced the included "eucarc" file to set up your Eucalyptus environment.

```
:$  . ~/.euca/eucarc
```

It is possible to add this command to your ~/.bashrc file so that the Eucalyptus environment is set up automatically when a user logs in. Eucalyptus treats this set of credentials as "administrator" credentials that allow the holder global privileges across the cloud. As such, they should be protected in the same way that other elevated-priority access is protected (e.g., should not be made visible to the general user population).

```
:$ echo "[ -r ~/.euca/eucarc ] && . ~/.euca/eucarc" >>
~/.bashrc
```

We installed the required cloud user tools:

```
:$ sudo apt-get install euca2ools
```

And validated that everything was working correctly by getting the local cluster availability details:

```
:$  . ~/.euca/eucarc euca-describe-availability-zones
verbose
```

Output: the cluster information.

Node controller (NC) Installation and Configuration

Server Information:

- Server name: NC1a
- IP Address: 192.168.32.2 – obtained from the associated Cluster Controller

Eucalyptus Node Controller Installation
The following steps must be repeated on each Node Controller in the cloud.

We installed the Eucalyptus-nc package.

```
:$ sudo apt-get install eucalyptus-nc
```

We set up the Node Controller to use an Ethernet bridge to the local network. The following script should configure a bridge correctly in many installations.

```
interface=eth0
bridge=br0
sudo sed -i "s/^iface $interface inet \(.*\)$/iface
$interface inet manual\n\nauto br0\niface $bridge inet
\1/" /etc/network/interfaces
sudo tee -a /etc/network/interfaces <<EOF
        bridge_ports $interface
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
EOF
:$ sudo /etc/init.d/networking restart
```

We configured /etc/eucalyptus/eucalyptus.conf with the name of the bridge, and restarted the node controller with the script shown below.

```
sudo sed -i "s/^VNET_BRIDGE=.*$/VNET_BRIDGE=$bridge/"
/etc/eucalyptus/eucalyptus.conf
sudo /etc/init.d/eucalyptus-nc restart
```

Install the right access for user Eucalyptus

To ensure that we have access to the cloud controller machine we need to do the following. Install the Cluster Controller's Eucalyptus user's public ssh key into the Node Controller's Eucalyptus user's authorized_keys file.

On the Node Controller, we temporarily set a password for the Eucalyptus user:

```
:$ sudo passwd eucalyptus
```

Then, on the Cluster Controller:

```
:$ sudo -u eucalyptus ssh-copy-id -i ~eucalyptus/.ssh/
id_rsa.pub eucalyptus@<IP_OF_NODE>
```

We removed the password of the Eucalyptus account on the Node:

```
:$ sudo passwd -d eucalyptus
```

Register nodes from the master cluster

```
:$ sudo euca_conf --no-rsync --register-nodes
<IP_OF_NODE>
```

To verify that node registration was successful, we checked for the existence of the /var/log/eucalyptus/nc.log file on NC1a.

We tested that the nodes were up and running:

```
:$ euca-describe-availability-zones verbose
```

To verify the system policies and configuration, we connected to the web interface: https://<CLC IP>:8443

Storage Server Installation

System Configuration:

- Server name: SM
- IP Address: 192.168.16.10

Storage Server Configuration

We installed the iscsi target using following command:

```
:$ sudo apt-get install iscsitarget
```

Edited /etc/ietd.conf:

```
:$ sudo vi /etc/ietd.conf
```

Added a rule to allow access to the initiator host:

```
:$  vi /etc/initiators.allow
```

Added the rule "ALL ALL" in /etc/initiators.deny:

```
:$ vi  /etc/initiators.deny
```

Finally, we restarted the iscsi target:

```
:$ /etc/init.d/iscsitarget restart
```

Configure Walrus and Block Storage Controller as iSCSI clients.

We installed open-iscsi package:

```
:$ sudo apt-get install open-iscsi
```

And restarted the service:

```
:$ /etc/init.d/open-iscsi restart
```

Note: These steps were required for both Walrus and Block Storage Controller.

Discovering the target:

We discovered target (has to be done only once per new target host).

```
:$ iscsiadm -m discovery -t sendtargets -p <ip of
target host>
:$ /etc/init.d/open-iscsi restart
```

Block device should appear under /dev (check dmesg).

```
:$ fdisk /dev/sdb and create a new partition.
:$ mkfs.ext3 /dev/sdb1
```

We mounted it. Let's call the mount point /mnt/disk0
Edit /etc/fstab and set up /dev/sdb1 to mount at boot.

```
:$ vi /etc/fstab
```

Depending on whether the host is Walrus or a Storage Controller, create a symbolic link:

```
ln -sf /var/lib/eucalyptus/bukkits /mnt/disk0
```

(or ln -sf /var/lib/eucalyptus/volumes /mnt/disk0 on the Storage Controller)

IMPORTANT: User "Eucalyptus" requires read/write permissions to the mounted disk.

Install UEC image from the store

The simplest way to add an image to UEC is to install it from the Image Store on the UEC web interface.

1. Access the Cloud Controller web interface. (Make sure you specify https.) https://<CLC IP>:8443
2. Enter your login and password. Click on the Store tab (Figure 5).
3. Browse available images.
4. Click on install for the image you want.
5. Verify the downloaded image (Figure 6).

Running the image

There are multiple ways to instantiate an image in UEC:

- Command line
- One of the UEC-compatible management tools such as Landscape
- ElasticFox extension to Firefox

In this example we will cover the command-line option.

Before running an instance of your image, first create a keypair (ssh key) that can be used to log into your instance as root, once it boots. You will only have to do this once as the key is stored. Run the following commands.

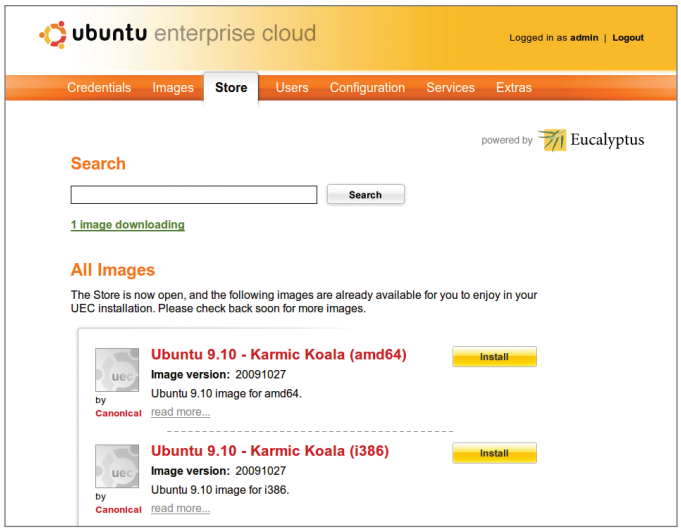


Figure 5. Screen shot of the UEC Image Store interface.

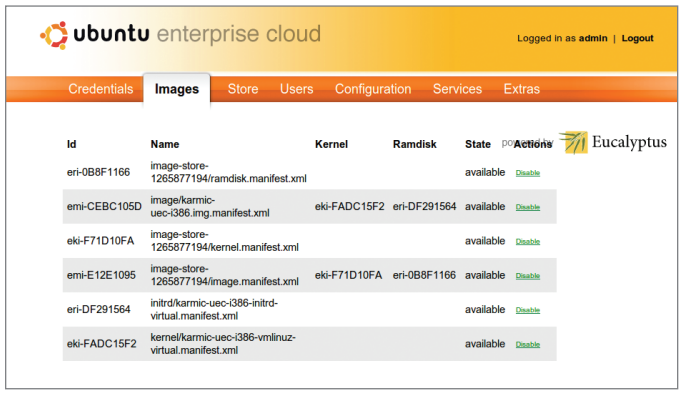


Figure 6. Screen shot showing registered images available to users.

```
:$  if [ ! -e ~/.euca/mykey.priv ]
touch ~/.euca/mykey.priv
chmod 0600 ~/.euca/mykey.priv
euca-add-keypair mykey > ~/.euca/mykey.priv
fi

ubuntu@CLC:~$ if [  ! -e ~/.euca/kamalkey.priv ]; then
> touch ~/.euca/kamalkey.priv
> chmod 0600 ~/.euca/kamalkey.priv
>euca-add-keypair kamalkey > ~/.euca/kamalkey.priv
> fi
ubuntu@CLC:~$ euca-describe-keypairs
KEYPAIR mykey   9f:b9:6c:d7: 32:a2:22:3d:96:74:0d:38:ab:d8:
6f:af:bf:b8:00:cf
KEYPAIR kamalkey        d1:09:cb:df:15:4b:7d:94:05:56:f8:7c
:92:a8:ea:36:67:64:1d:71
KEYPAIR paulkey d8:2a:67:31:b3:a4:d2:89:95:d5:63:c5:1e:c5:
e5:e7:f9:71:c3:d8
KEYPAIR neil-key        59:78:ad:3b:89:ce:a3:3e:52:4d:c1:a
8:96:87:84:24:33:ba:03:6b
ubuntu@CLC:~$ _
```

We allowed access to port 22 in the instances:

```
:$ euca-describe-groups
:$ euca-authorize default -P tcp -p 22 -s 0.0.0.0/0
```

Listed the images.

```
:$ euca-describe-images
```

And ran the instances of the registered image.

```
:$ euca-run-instances $EMI -k mykey -t c1.medium
```

(EMI stands for Eucalyptus Machine Image)

$EMI is taken from euca-describe-images, you can save it in your .bashrc

```
ubuntu@CLC :~$ euca-run-instances emi-DCD31053 -k
kamalkey - t c1. medium
RESERVATION   r-2B580630 admin       admin-default
INSTANCE      i-55450A07 emi-DCD31053     0.0.0.0 0.0.0.0
pending kamalkey
2009-12-22T00:11:58.869Z  eki-F2F610D2  eri-0769113B
```

```
:$ watch -n 2 euca-describe-instances
```

Get the IP from running instances and ssh by examining the results from the euca-describe-instances results.

```
:$ ssh -i ~/.euca/mykey.priv ubuntu@$IP
```

Use Case Details

To demonstrate the capabilities of the cloud, we implemented the following use cases with the focus on the IaaS stack.

Key actors for these use cases are:

1. Service Provider (SP)
2. Service Consumers (SC1 and SC2)

Pre-conditions:

1. The cloud management software, UEC software is installed and ready to go.
2. Compute and storage nodes are installed and registered.
3. SC1 and SC2 application services (Service 1 and Service 2) are packaged in VMs to be deployed on the compute nodes.

Use Cases:

1. SP: Create two users (SC1, SC2) via the admin portal: Using the self-service portal, create two users to mimic two customers accessing the cloud.
2. SC1: Create instance of the Service 1: Instantiate virtual machine's that make up the Service 1 including IP address and links to storage.
3. SC1: Monitor state of Service 1: Observe the state of the newly created VMs.
4. SC1: Scale-out Service 1: Add an app front-end VMs and add to load-balance pool. Test for application scalability.
5. SC2: Create instance of the Service 2: Instantiate VMs that make up the Service 2 including IP address and links to storage.
6. SC2: Monitor state of Service 2: Observe the state of the newly created VMs.
7. SC1: Terminate an app front-end VM: Remove from load-balance pool and terminate a VM and observe for results.
8. SP: Add bare-metal capacity to existing cluster: Add a new server to an existing cluster.
9. SC1, SC2: Generate utilization reports: End users requesting for resource utilization report of their usage.
10. SP: Generate utilization reports: Create reports either via a GUI or by log files.
    - SP: Balance utilization for power (or other) metrics and adjust workload placement policy.
11. SC1, SC2: Scale-out Service 1, Service 2: Add an app front-end VM and add to load-balance pool. Test for application scalability on both Service 1 and Service 2.
12. SP: Fail a server; Ensure that the cluster is still operational.
13. SC1, SC2: Shutdown Service 1, Service 2: End the application service and remote access to users SC1 and SC2.
14. SP: Generate utilization reports: Create resource utilization report of SC1 and SC2.

Execution of Use Cases and Results

Compute and Storage Nodes installed and registered

SSH to CLC and pass the below command to list the number of Walrus, Clusters, and Storage controllers.

```
ubuntu@CLC:~$  sudo euca_conf --list-walruses
[sudo] password for ubuntu:
registered walruses:
   walrus 192.168.16.3
ubuntu@CLC:~$  sudo euca_conf --list-clusters
registered clusters:
   CCa 192.168.16.4
   CCb 192.168.16.5
ubuntu@CLC:~$  sudo euca_conf --list-scs
registered storage controllers:
   CCa 192.168.16.4
   CCb 192.168.16.5
ubuntu@CLC:~$ _
```

List all the resources

SSH to CLC and pass the below commands to list the availability zones. This report shows the running instances in the various availability zones.

```
:$ sudo euca-describe-availability-zones verbose
```

```
ubuntu@CLC:~$ euca-describe-availability-zones verbose
AVAILABILITYZONE  CCa    192.168.16.4
AVAILABILITYZONE  |- vm types   free / max  cpu ram  disk
AVAILABILITYZONE  |- m1.small   0064 / 0064  1   128    2
AVAILABILITYZONE  |- c1.medium  0064 / 0064  1   256    5
AVAILABILITYZONE  |- m1.large   0032 / 0032  2   512    10
AVAILABILITYZONE  |- m1.xlarge  0032 / 0032  2  1024    20
AVAILABILITYZONE  |- c1.xlarge  0016 / 0016  4  2048    20
AVAILABILITYZONE  CCb    192.168.16.5
AVAILABILITYZONE  |- vm types   free / max  cpu ram  disk
AVAILABILITYZONE  |- m1.small   0064 / 0064  1   128    2
AVAILABILITYZONE  |- c1.medium  0064 / 0064  1   256    5
AVAILABILITYZONE  |- m1.large   0032 / 0032  2   512    10
AVAILABILITYZONE  |- m1.xlarge  0032 / 0032  2  1024    20
AVAILABILITYZONE  |- c1.xlarge  0016 / 0016  4  2048    20
ubuntu@CLC:~$
```

Create service instances

This test instantiates 20 instances simultaneously.

SSH to CLC and pass below commands to initiate the 20 instances simultaneously.

```
:$ euca-run-instances –n 20 emi-DCD31053 –k paulkey –t
c1.medium
```

On the first time this command is used, it may take some to time for the instances to be in the "running" state..

Monitor state of service (resource distribution)

SSH to CLC and initiate 20 instances and pass below commands to check the resources distribution across the clusters. In the screen shot below, the running instances should be split evenly across the clusters.

```
:$ euca-describe-availability-zones verbose
```

Check the resources retaining after the instances are terminated

To terminate the running instances use the below command sequence.

```
:$ euca-terminate-instances <instance id>
```

Create two security groups and monitor VMs

Creating security groups will allow groups of images to work on different sealed networks:

```
:$ euca-add-group -d "Group Client 3" client_3
:$ euca-authorize client_1 -P tcp -p 22 -s 0.0.0.0/0
:$ euca-run-instances $EMI -k mykey -t c1.medium -g
client_3
:$ euca-run-instances $EMI -k mykey -t c1.medium -g
client_3
:$ euca-add-group -d "Group Client 4" client_4
:$ euca-authorize client_2 -P tcp -p 22 -s 0.0.0.0/0
:$ euca-run-instances $EMI -k mykey -t c1.medium -g
client_4
:$ euca-run-instances $EMI -k mykey -t c1.medium -g
client_4
:$ euca-describe-instances
```

In this case, client_3 has two instances running with IP (192.168.18.1 and 192.168.17.1). Client_4 has two instances running with IP (192.168.18.10 and 192.168.17.10).

```
ubuntu@CLC:~$ euca-describe-instances

RESERVATION  r-47480905   admin      client_3
INSTANCE     i-34DB0608   emi-DCD31053 192.168.18.1
10.0.9.3 pending     kamalkey 0     c1.medium
2009-12-22T02:02:35.707Z
CCb        eki-F2F610D2       eri-0769113B
RESERVATION  r-46620893   admin     client_3
INSTANCE     i-429E0823   emi-DCD31053 192.168.17.1
10.0.9.2
running     kamalkey     0    c1.medium
2009-12-22T02:02:13.876Z
CCa        eki-F2F610D2       eri-0769113B
RESERVATION  r-33D406B6   admin     client_4
INSTANCE     i-50E80959   emi-DCD31053 192.168.18.10
10.0.10.3
pending     kamalkey     0    c1.medium
2009-12-22T02:03:34.748Z
CCb        eki-F2F610D2   eri-0769113B
RESERVATION  r-55030911   admin     client_4
INSTANCE     i-45F60790   emi-DCD31053 192.168.17.10
10.0.10.2
running     kamalkey     0    c1.medium
2009-12-22T02:03:28.757Z
CCa        eki-F2F610D2   eri-0769113B
ubuntu@CLC:~$  _
```

Check the network isolation between two clusters

SSH to the one of the instances and try to ping CC,CLC, S3.

```
:$ ssh –i .euca/<key.priv> <instance IP>
ubuntu@10:~$ ping   192.168.16.2
PING 192.168.16.2  (192.168.16.2) 56(84) bytes of data.
64 bytes from 192.168.16.2: icmp _ seq=1 ttl=63
time=0.405 ms
64 bytes from 192.168.16.2: icmp _ seq=2 ttl=63
time=0.248 ms
^C
--- 192.168.16.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss,
time 999ms
rtt min/avg/max/mdev = 0.248/0.326/0.405/0.080 ms
ubuntu@10:~$ ping   192.168.16.4
PING 192.168.16.4  (192.168.16.4) 56(84) bytes of data.
64 bytes from 192.168.16.4: icmp _ seq=1 ttl=63
time=0.338 ms
^C
--- 192.168.16.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss,
time 0ms
rtt min/avg/max/mdev = 0.338/0.338/0.338/0.000 ms
ubuntu@10:~$ ping   192.168.16.5
PING 192.168.16.5  (192.168.16.5) 56(84) bytes of data.
64 bytes from 192.168.16.5: icmp _ seq=1 ttl=64
time=0.278 ms
^C
--- 192.168.16.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss,
time 0ms
rtt min/avg/max/mdev = 0.278/0.278/0.278/0.000 ms
ubuntu@10:~$
```

Application Scalability

The goal of this use case is to test the deployment of a custom image proposing an auto-scaling web service. The web service is in fact a single web page calculating a fractal image, but the various requests are balanced over a pool of virtual servers.

In preparation for this test, we created a VM comprised of an Ubuntu server, Apache, and a PHP application for computing the Mandelbrot fractal. The steps below use that VM image as the input to bundle the image for use in our cloud.

Image preparation: kernel, ramdisk and image

```
:$ euca-bundle-image –i fractal _ vmlinuz –r x86 _ 64
--kernel true
:$ euca-upload-bundle –b k-scal –m /tmp/fractal _ vmli-
nuz.manifest.xml
:$ euca-register k-scal/fractal _ vmlinuz.manifest.xml
#note the resulting ID
:$ euca-bundle-image –i fractal _ initrd –r x86 _ 64
--ramdisk true
:$ euca-upload-bundle –b r-scal –m /tmp/fractal _
initrd.manifest.xml
:$ euca-register r-scal/fractal _ initrd.manifest.xml
#note the resulting ID
:$ euca-bundle-image –i fractal _ img –r x86 _ 64
--kernel eki-82650F40 --ramdisk eri-6C020EAC
:$ euca-upload-bundle –b i-scal –m /tmp/fractal _ img.
manifest.xml
:$ euca-register i-scal/fractal _ img.manifest.xml
#note the resulting ID
```

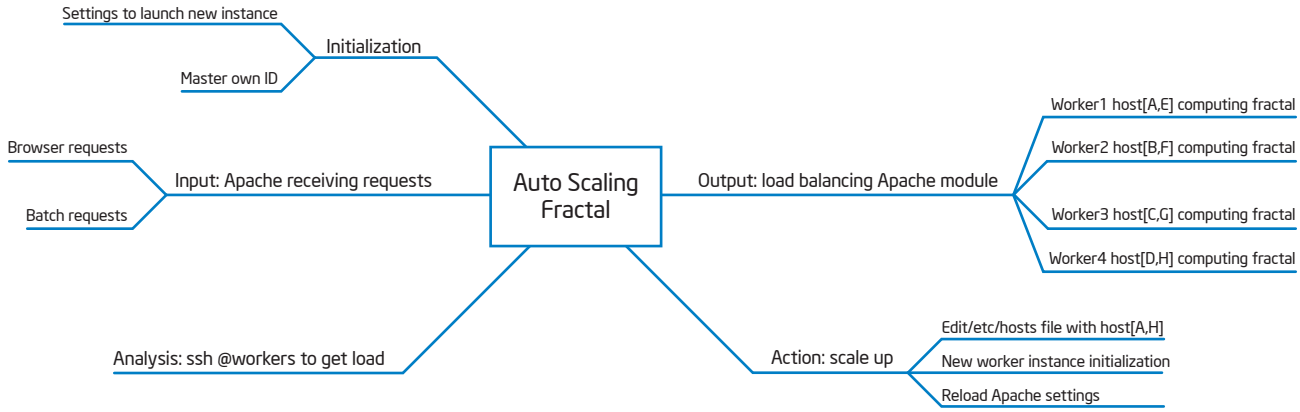


Figure 7. Example of auto-scaling web server application.

Launching 1 master and 1 worker instances, ssh to the master

```
:$ euca-run-instances -k mykey -n 2 -z CCa emi-68270E9C
-t m1.large # replace with your image ID
:$ euca-describe-instances | grep emi-68270E9C #
replace with your image ID


INSTANCE        i-44130714      emi-68270E9C
192.168.17.1    10.0.8.2        running         mykey
0       m1.large 2010-01-08T14:24:08.688Z         CCa
eki-B89F1068    eri-A2BA0FDC
INSTANCE        i-4A1308D3      emi-68270E9C
192.168.17.10   10.0.8.3        running         mykey
1       m1.large 2010-01-08T14:24:08.688Z         CCa
eki-B89F1068    eri-A2BA0FDC
```

Copy files to the cluster controller: If CCa is the Cluster Controller of the cluster where the first two instances where launched:

```
:$ scp -r /home/ubuntu/.euca/ CCa:~/euca-credentials
:$ scp -r scripts-master CCa:~/scripts-master/
```

You need to be able to ssh to the instance and make a web request (port 80), so make sure you have the right security rights:

```
:$ euca-describe groups
GROUP   admin   default default group
PERMISSION      admin   default ALLOWS  tcp     22
22      FROM    CIDR    0.0.0.0/0
PERMISSION      admin   default ALLOWS  tcp     80
80      FROM    CIDR    0.0.0.0/0
```

Select one of the two instances as the master of your fractal load balancer; we picked 10.0.8.2 i-44130714. Note that our fractal image is using the root user account directly (that's unusual for Ubuntu, so remember to use "root@"):

```
:$ scp -i ~/.euca/mykey.priv -r ~/.euca root@10.0.8.2:~/
euca-credentials
:$ scp -i ~/.euca/mykey.priv -r scripts-master
root@10.0.8.2:~/scripts-master
:$ ssh -i ~/.euca/mykey.priv root@10.0.8.2
:$ cd euca-credentials
```

Master configuration and launch

Create a file called "balance.conf" and add the value BALANCER_EMI and LOCAL_ID.

```
:$ echo "export BALANCER _ EMI=emi-68270E9C" > balance.
conf
:$ echo "export LOCAL _ ID=i-44130714" >> balance.conf
:$ cd /root/scripts-master/ ./euca-balancer-init.sh
```

A few things are hard coded in euca-balancer.pl, like the cluster name "CCa", or the keyname. So, if you use different settings then you will need to edit the perl script.

```
emi-68270E9C, i-44130714
10.0.8.3        HOSTA
10.0.8.3        HOSTB
10.0.8.3        HOSTC
10.0.8.3        HOSTD
10.0.8.3        HOSTE
10.0.8.3        HOSTF
10.0.8.3        HOSTG
10.0.8.3        HOSTH
127.0.0.1       localhost.localdomain localhost
* Reloading web server config apache2
[ OK ]
LOAD: 10.0.8.3 = 0.01
AVG: 0.01
```

Looking at the output above

- The image reference and master instance information is at the top
- Then a list of worker IP addresses able to receive http request. By default the eight worker slots are all pointing to the only worker we've launched
- Apache is reloaded for each script cycle, reading the new worker IP list. Apache on the master is just load balancing requests for workers
- The loads of all worker is shown

First load

All we have to do is send requests (from the browser) to the load balancer at this address: http://192.168.17.1/balanced/fractal.php.

You should see a fractal image. The image creation will increase the worker image load. The script will detect it, launching a new worker, creating a new host file with the two workers. The Apache configuration will be reloaded and future requests sent to the two workers.

Power Footprint

Power optimization is an important consideration in cloud implementations. Not only because of the higher density but also because reducing the power footprint of a data center reduces the overall operation cost. One of the simplest forms of power management is to put servers into a lower power mode when they are idle. UEC implements a simple form of this power management using a Canonical-sponsored tool known as PowerNap. PowerNap puts a server into sleep mode when it determines that the server is no longer being utilized. Then, when UEC needs the server, UEC uses Wake-on-LAN to wake up the server so that workloads can subsequently be placed on the server.

One of the factors is the time to put the server to sleep and to wake the server. Below are the measurements we made on the servers in our clusters. The time to hibernate and resume will vary depending on CPU, the amount of memory, and disk performance.

Scenario	Time
Time to hibernate the node	1:00 (1 minute)
Time to resume a node from hibernate	3:08
Time to power-on the node using wake-up command (boot OS without hibernation)	2:40

Based on our particular hardware configuration, it is likely that we will see the best recovery times by simply powering off the nodes and starting from fresh rather than using hibernation or sleep modes.

Prerequisites:

1. Your node(s) must support Wake-on-LAN, and have it enabled in the BIOS.
2. To get PowerNap, do the following.

```
:$ sudo apt-get install powernap
```

You run PowerNap on each node that participates in power management. Edit the PowerNap configuration file to indicate the priority of actions.:* /etc/powernap/action.

```
* pm-suspend
* pm-hibernate
* poweroff
```

The node should be able to execute at least one of these.

To enable UEC's Power Management, enable the POWERSAVE scheduling algorithm in your Cloud Controller's /etc/eucalyptus/eucalyptus.conf. (You will need to do this on all CCs.)

```
: $ sudo sed -i 's/^SCHEDPOLICY=.*$/
SCHEDPOLICY="POWERSAVE"/' /etc/eucalyptus.conf
:$ sudo restart eucalyptus CLEAN=1
```

By default, Eucalyptus will wait for 300 seconds (5 minutes) of inactivity on a node before putting it to sleep. You can monitor the state by watching the cc.log:

```
:$ tail -f /var/log/eucalyptus/cc.log
```

Obtain admin account credentials through Web Interface

Connect to Cloud Controller using a Web browser to intialize login credentials and obtain the admin account certificate.

1. From a Web browser, access the following URL: https://<cloud-controller-ip-address>:8443/
2. Use username 'admin' and password 'admin' for the first time login.
3. Then follow the on-screen instructions to update the admin password and e-mail address.
4. Once the first-time configuration process is completed, click the "credentials" tab located in the top-left portion of the screen.
5. Click the "Download Credentials" button to get your certificates.
6. Save them to ~/.euca.
7. Unzip the downloaded zipfile into a safe location (~/.euca).

```
unzip -d ~/.euca mycreds.zip
```

Other users that are not admin must get their credentials through the Web interface. Web interface can also be used by administrators to validate the users.

Things to Consider

Use a SAN Rather Than an iSCSI Server?

We chose to use a Storage Server with local DAS for the bulk storage requirements in our cloud. We could just as well have connected the Storage Server to a SAN. The choice in this case was completely arbitrary based on the capabilities in our lab setup – we already had a server with local storage (DAS, JBOD).

The more fundamental requirement is for the Node Controllers to have access to a shared storage device with sufficient capacity and performance. A NAS device connected to a SAN would have worked just as well. The choice is really driven by design factors such as the back-up strategy, performance, and cost.

Scalability

Consider the case where the cloud we've described has been implemented and is in use. But now, there is demand for a considerably larger cloud. Below are considerations when increasing the capacity of the cloud.

If the reason to increse the size of the cloud is to get more compute power, then just add servers to the existing clusters. If the need is for improved availability, then more clusters are needed, which in turn, creates new availability zones. With the additional availability zones, applications can be spread out so that the failure of a Cluster Controller or a switch will not impact the application.

Increasing the scale of the cloud may have other side effects as well. Below are some factors to consider as the size of the cloud is increased.

Network capacity:

- As additional applications are added, the network traffic will increase. The increase will come not only from the application traffic but also for any storage activities.
- Network traffic will increase if there is a higher load on the applications (e.g., maintenance load for DR and data replication) that causes an increased number of users or changing access patterns.

Cloud Topology:

- Consider how the cluster/node topology will influence robustness and/or performance of the cloud solution. Consider how network faults will impact the user-perceived performance of the cloud. Note that the failure of a switch not only breaks the network connections but will often leave a segment of the network isolated for a period of time.
- Storage topology is also a key consideration as a separate data network and storage solution can limit scale benefits. Be sure to look at the performance demands that will be placed on the Block Storage Controllers and the networks that connect to them.

Consider the case of needing to add support for 1,000 virtual machines to the cloud.

- Eucalyptus supports up to eight clusters in a single cloud. Based on analysis of workloads, one might come to a design point where we average 16 "small" VM instances per server. Note that many servers will have more VM instances based on server performance and workload demands. Using the above average, this would mean that we need:
  - 1000/16 = 63 Node Controllers (i.e., servers) total in the cloud to support the incremental 1,000 VMs.
  - Assuming we have the maximum number of clusters (eight) then we have eight Node Controllers per Cluster.
  - Or, assuming we stay with the Class C IP address space, we are limited to 10 Node Controllers per Cluster, we would need seven Clusters.

Would the Use of SSD Drives Improve Performance?

Using SSDs as hard disk replacements can improve performance in a cloud. However, to get the best use of the SSDs, there are two locations in the cloud that can especially benefit.

- The Node Controllers use the local disk to cache VM images. Therefore, if many copies of the same instance will be run on the same node, using an SSD as the storage device for a Node Controller can greatly speed up the loading of VMs.
- The Walrus Storage Service is an object store. This means that for each request for an object, the location of the object must be determined from the provided key. This lookup operation is done frequently from a metadata store. Using SSDs to hold this metadata store can greatly improve the performance for the process of locating the object.

Appendix

Comments were removed from the following for brevity.

Cluster Controller .conf file

```
EUCALYPTUS="/"
EUCA_USER="eucalyptus"
DISABLE_DNS="Y"
ENABLE_WS_SECURITY="Y"
LOGLEVEL="DEBUG"
CC_PORT="8774"
SCHEDPOLICY="POWERSAVE"
POWER_IDLETHRESH="300"
POWER_WAKETHRESH="300"
NODES=" NC1a NC2a NC3a NC4a"
NC_SERVICE="axis2/services/eucalyptusNC"
NC_PORT="8775"
HYPERVISOR="not_configured"
INSTANCE_PATH="not_configured"
VNET_PUBINTERFACE="eth0"
VNET_PRIVINTERFACE="br1"
VNET_BRIDGE="br0"
VNET_DHCPDAEMON="/usr/sbin/dhcpd3"
VNET_DHCPUSER="dhcpd"
VNET_MODE="MANAGED-NOVLAN"
VNET_SUBNET="10.0.0.0"
VNET_NETMASK="255.0.0.0"
VNET_DNS="192.168.16.11"
VNET_ADDRSPERNET="256"
VNET_PUBLICIPS="192.168.17.1-192.168.17.254"
VNET_CLOUDIP="192.168.16.2"
```

Cloud Controller .conf file

```
EUCALYPTUS="/"
DISABLE_DNS="Y"
ENABLE_WS_SECURITY="Y"
LOGLEVEL="DEBUG"
CC_PORT="8774"
SCHEDPOLICY="ROUNDROBIN"
POWER_IDLETHRESH="300"
POWER_WAKETHRESH="300"
NODES=""
NC_SERVICE="axis2/services/eucalyptusNC"
NC_PORT="8775"
HYPERVISOR="not_configured"
INSTANCE_PATH="not_configured"
VNET_PUBINTERFACE="eth0"
VNET_PRIVINTERFACE="eth0"
VNET_BRIDGE="br0"
VNET_DHCPDAEMON="/usr/sbin/dhcpd3"
VNET_DHCPUSER="dhcpd"
VNET_MODE="MANAGED-NOVLAN"
```

Node Controller .conf file

```
EUCALYPTUS="/"
DISABLE_DNS="Y"
ENABLE_WS_SECURITY="Y"
LOGLEVEL="DEBUG"
CC_PORT="8774"
SCHEDPOLICY="ROUNDROBIN"
POWER_IDLETHRESH="300"
POWER_WAKETHRESH="300"
NODES=""
NC_SERVICE="axis2/services/eucalyptusNC"
NC_PORT="8775"
HYPERVISOR="kvm"
INSTANCE_PATH="/var/lib/eucalyptus/instances"
VNET_PUBINTERFACE="eth0"
VNET_PRIVINTERFACE="eth0"
VNET_BRIDGE="br0"
VNET_DHCPDAEMON="/usr/sbin/dhcpd3"
VNET_DHCPUSER="dhcpd"
VNET_MODE="MANAGED-NOVLAN"
```

Glossary

AWS: Amazon Web Service.

Block Storage Controller (SC): Eucalyptus component that manages dynamic block storage services (EBS). Each "cluster" in a Eucalyptus installation can have its own Block Storage Controller. This component is provided by the "eucalyptus-sc" package.

Cloud Controller (CLC): Eucalyptus component that provides the web UI (an https server on port 8443), and implements the Amazon EC2 API. There should be only one Cloud Controller in an installation of UEC. This service is provided by the Ubuntu eucalyptus-cloud package.

Cluster: A collection of nodes, associated with a Cluster Controller. There can be more than one Cluster in an installation of UEC. Clusters are sometimes physically separate sets of nodes. (e.g., floor1, floor2, floor2).

Cluster Controller (CC): Eucalyptus component that manages collections of node resources. This service is provided by the Ubuntu eucalyptus-cc package.

DCM: Data Center Manager. DCM is a package from Intel to provide policy based tools for managing power in the data center.

DCMI: Data Center Manageability Interface: Datacenter Manager Specifications are derived from Intelligent Platform Management Interface (IPMI) 2.0, which has been widely adopted by the computing industry for server management and system-health monitoring. The Datacenter Manager specifications define a uniform set of monitoring, control features and interfaces that target the common and fundamental hardware management needs of server systems that are used in large deployments within data centers, such as Internet Portal data centers. This includes capabilities such as secure power and reset control, temperature monitoring, event logging, and others.

EBS: Elastic Block Storage.

EC2: Elastic Compute Cloud. Amazon's pay-by-the-hour public cloud computing offering.

EKI: Eucalyptus Kernel Image.

EMI: Eucalyptus Machine Image.

ERI: Eucalyptus Ramdisk Image.

Eucalyptus: Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems. An open-source project originally from the University of California at Santa Barbara, now supported by Eucalyptus Systems.

Infrastructure as a Service (IaaS): The delivery of computer infrastructure (typically a platform virtualization environment) as a service (From Wikipedia).

KVM: Kernel Virtual Machine. A hypervisor embedded in the Linux kernel.

Node: A node is a physical machine that's capable of running virtual machines, running a node controller. Within Ubuntu, this generally means that the CPU has VT extensions, and can run the KVM hypervisor.

Node Controller (NC): Eucalyptus component that runs on nodes which host the virtual machines that comprise the cloud. This service is provided by the Ubuntu package eucalyptus-nc.

Power Manager: The function that is responsible for managing the power utilization in the cloud. DCM was used for the Power Manager.

S3: Simple Storage Service. Amazon's pay-by-the-gigabyte persistent storage solution for EC2.

Ubuntu: A Linux distribution sponsored by Canonical.

UEC: Ubuntu Enterprise Cloud. Ubuntu's cloud computing solution, based on Eucalyptus.

VM: Virtual Machine.

VT: Virtualization Technology. An optional feature of some modern CPUs, allowing for accelerated virtual machine hosting.

Walrus: Eucalyptus component that implements the Amazon S3 API, used for storing VM images and user storage using S3 bucket put/get abstractions.

References

1. Why the Intel® Xeon® Processor 5500 Series is the Ideal Foundation for Cloud Computing, http://communities.intel.com/docs/DOC-4213.

2. Intel Xeon Processor 5500 Series Software Industry Testimonials, http://www.intel.com/business/software/testimonials/xeon5500.htm.

3. Intel Virtualization Technology, http://www.intel.com/technology/virtualization/, http://download.intel.com/business/resources/briefs/xeon5500/xeon_5500_virtualization.pdf.

4. Baidu POC, http://software.intel.com/sites/datacentermanager/intel_node_manager_v2e.pdf.

5. Intel in cloud computing Wiki, http://communities.intel.com/docs/DOC-4230.

6. Intel® Cloud Builder Program, http://communities.intel.com/docs/DOC-4292.

7. Hadoop, http://hadoop.apache.org/.

8. Amazon Web Services, http://aws.amazon.com.

9. Eucalyptus Administrator Guide, http://open.eucalyptus.com/wiki/EucalyptusAdministratorGuide.

10. Ubuntu Enterprise Cloud web site, http://www.ubuntu.com/cloud.

11. UEC installation and administration guide, http://help.ubuntu.com/community/UEC.

12. Squid deb proxy documentation, https://wiki.ubuntu.com/SquidDebProxy.

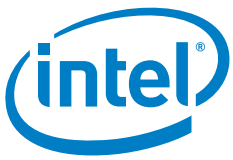
13. Data Center Manager, http://software.intel.com/sites/datacentermanager/.

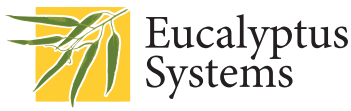
14. Cloud Storage Power Management Reference Architecture, http://communities.intel.com/servlet/JiveServlet/downloadBody/4316-102-3-7063/CloudPowerMgt1.02.pdf.

15. Intel® Xeon® processor 5500 series-based documentation, http://ark.intel.com/ProductCollection.aspx?codeName=33163, http://www.intel.com/support/processors/xeon5k/.

16. DCMI Specification, http://www.intel.com/technology/product/DCMI/index.htm.

To learn more about deployment of cloud solutions,
visit www.intel.com/software/cloudbuilder





Printed in USA 0310/RH/OCG/XX/PDF Please Recycle 323488-001US