OSDC 2012

24th April. Nürnberg

Building Clouds with OpenNebula 3.2

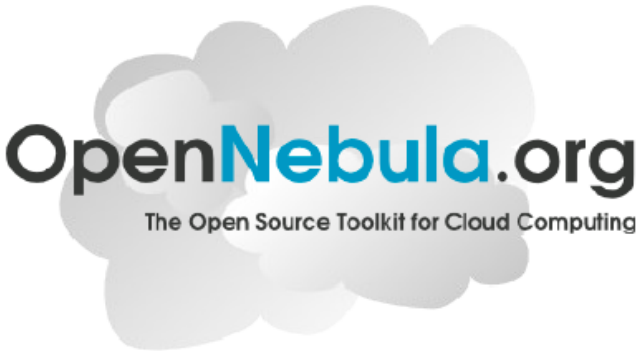
Constantino Vázquez Blanco

dsa-research.org | OpenNebula.org

Distributed Systems Architecture Research Group
Universidad Complutense de Madrid

Building Clouds with OpenNebula 3.4
Public Cloud Computing

Constantino Vázquez Blanco
dsa-research.org | OpenNebula.org
Distributed Systems Architecture Research Group
Universidad Complutense de Madrid

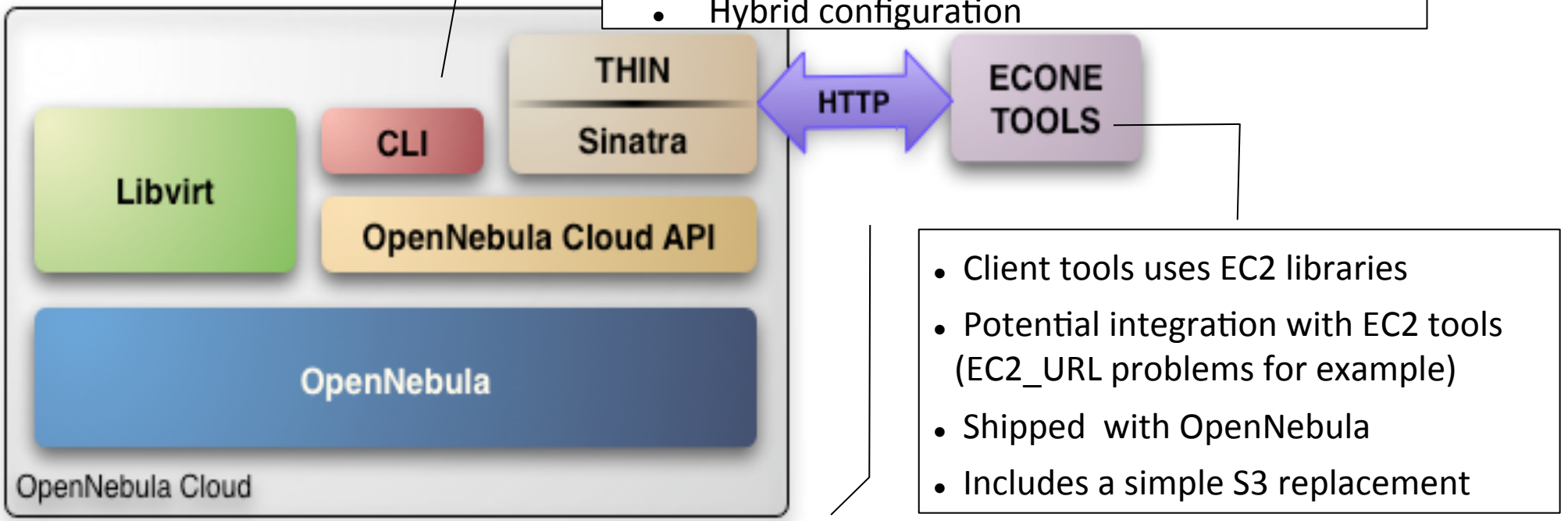


- Public Cloud Computing with OpenNebula
- Installing a Public Cloud with EC2 API
- Configuring the Public Cloud
- Using the Public Cloud (EC2)
- Overview and Hands-On OCCI
- OpenNebula Self-service

Public Cloud Computing with OpenNebula

Overview

- You can use multiple interfaces for the Cloud
- Transparent to your setup:
  - Hypervisor
  - Storage Model
  - Hybrid configuration



- Client tools uses EC2 libraries
- Potential integration with EC2 tools (EC2_URL problems for example)
- Shipped with OpenNebula
- Includes a simple S3 replacement

- Supports HTTP and HTTPS protocols
- *EC2 authentication* based on OpenNebula credentials
- Public Cloud users need an OpenNebula account

Installing the Public Cloud

Runtime Requirements (front-end)

- OpenNebula distribution supports two Cloud interfaces: the EC2 Query API and OCCI
- Additional requirements: EC2 development library, web server and web framework

```
# gem install amazon-ec2 uuid
# gem install sequel
# apt-get install curl libcurl3 libcurl4-gnutls-dev
# gem install curb
# gem install sqlite3-ruby

Add a "FQDN" for our Public Cloud
# vim /etc/hosts
127.0.0.1       localhost
#127.0.1.1      pcaulaX

193.144.33.y    pcaulaX pcaulaX.opennebula.org
```

Configuring the Public Cloud

Server Options and Instance types

- The EC2 service is configured in $ONE_LOCATION/etc/econe.conf

```
# OpenNebula sever contact information
ONE_XMLRPC=http://localhost:2633/RPC2

# Host and port where econe server will run
SERVER=pcaulaX.opennebula.org
PORT=4567

# SSL proxy that serves the API (set if is being used)
#SSL_SERVER=fqdm.of.the.server

# VM types allowed and its template file (inside templates directory)
VM_TYPE=[NAME=m1.small, TEMPLATE=m1.small.erb]
```

Configuring the Public Cloud

Define the Instances

- You have to define the correspondence between **types** (simple) and **local instantiation of VMs** (hard, you should be fine by now)
  - Capacity allocated by this VM type (CPU, MEMORY)
  - Your cloud requirements, e.g. force to use a given kernel (OS) or place public VMs in a given set of cluster nodes (REQUIREMENTS)
  - The network used by Public VMs (NIC)
- VM Types are **defined in econe.conf**. Templates for the VM templates are in $ONE_LOCATION/etc/ec2query_templates
- Templates for VM Types are erb files <% Ruby code here %>, you should not need to modify that.

Configuring the Public Cloud

Define the Instances

```
$ more m1.small.erb
NAME   = eco-vm

#Adjust Capacity for this instance type
CPU    = 0.1
MEMORY = 64

DISK   = [ IMAGE_ID = <%= erb_vm_info[:img_id] %> ]

NIC    = [ NETWORK_ID = 0 ]

IMAGE_ID      = <%= erb_vm_info[:ec2_img_id] %>
INSTANCE_TYPE = <%= erb_vm_info[:instance_type ]%>

<% if erb_vm_info[:user_data] %>
CONTEXT = [
       EC2_USER_DATA="<%= erb_vm_info[:user_data] %>",
       TARGET="hdc" ]
<% end %>
```

Configuring the Public Cloud

Start the EC2 Server

- **Hands on!**
  - Start the EC2 server
  - Adjust the m1.small template
  - Create additional “public” users with oneuser create

```
$ econe-server start

$ /usr/sbin/lsof -Pi

Check $ONE_LOCATION/var/econe-server.log for errors
```

Using the Public Cloud

The econe Toolset

- The **econe-tools** are a subset of the functionality provided by the onevm utility, and resembles the ec2-* cli
- EC2 ecosystem *can* be used (e.g. elasticfox, euca2ools…)
- **Image** related commands are:
  - **econe-upload**, place an image in the Cloud repo and returns ID
  - **econe-describe-images**, lists the images
  - econe-register, register an image
- **Instance** related commands are:
  - **econe-run-instances**, starts a VM using an image ID
  - **econe-describe-instances**, lists the VMs
  - **econe-terminate-instances**, shutdowns a VM

Using the Public Cloud

The econe Toolset

- **User authentication** is based in the OpenNebula credentials
  - AWSAccessKeyId is OpenNebula's username
  - AWSSecretAccessKey is OpenNebula's password
- Pass **your credentials** to the econe-tools by (in this order)
  - Command arguments (-K <username>, -S <pass>)
  - Environment EC2_ACCESS_KEY and EC2_SECRET_KEY
  - Environment ONE_AUTH
- Point econe-tools to your **target cloud**
  - Command arguments (-U <http|https>://<fqdn>:<port>) port needed if not the default for the protocol
  - EC2_URL environment

Example, Running a VM through the EC2 Interface

- **Hands on!**
  - Check the images in your cloud and start using it
  - Compare the **econe-*** (public view) and **one*** and **sunstone** (local view) evolution and information
  - Check the template build by the econe server (onevm show)
  - Upload the ttylinux image again and instance it

Using the Public Cloud

Example, Running a VM through the EC2 Interface

```
$ econe-upload -U http://node-x.opennebula.org:4567 --access-key ec2-
user --secret-key ec2-pass /srv/cloud/images/ttylinux/ttylinux.img
Success: ImageId ami-00000003

$ export EC2_URL=http://pcaulax.opennebula.org:4568
$ export EC2_ACCESS_KEY=ec2-user
$ export EC2_SECRET_KEY=ec2-pass

$ econe-describe-images -H
Owner        ImageId       Status          Visibility   Location
------------------------------------------------------------------
ec2-user     ami-00000003  available       private      23151fac850e5...

This is the local view not accessible to public cloud users
$ oneimage list
ID NAME                          TYPE  REGTIME      PUB PER STAT  #VMS
...
3 ec2-user ec2-71654e30-0872-01   OS   Jan 22, 2011  No  No  rdy     0

$ oneimage show 3
```

Configuring the Hybrid Cloud

Register the EC2 Cloud

```
$ econe-run-instances ami-00000003
ec2-user      ami-00000004    i-16           m1.small

$ econe-describe-instances -H
Owner         Id            ImageId        State         IP              Type
-------------------------------------------------------------------------------
ec2-user      i-10          ami-00000003   running       172.16.10.7   m1.small

This is the local view not accessible to public cloud users

$ onevm list
   ID     USER     NAME STAT CPU     MEM         HOSTNAME        TIME
    1  oneuser ttylinux runn  99    63.5M              n04 01 02:41:14
   10 ec2-user   eco-vm runn  99    63.8M              n04 00 01:05:28

$ onevm show 14
VIRTUAL MACHINE 14 INFORMATION
ID              : 14
NAME            : eco-vm
STATE           : ACTIVE
...
```

Configuring the Public Cloud

SSL Security to access the EC2 Server

- **SSL security is handle by a proxy** that forwards the request to the EC2 Query Service and takes back the answer to the client
- **Requirements**:
  - A server certificate for the SSL connections
  - An HTTP proxy that understands SSL
  - EC2Query Service configuration to accept petitions from the proxy

Configuring the Public Cloud

SSL Security to access the EC2 Server

- Install the proxy (lighttpd in our course)
- Generate the server certificates for your cloud
- Configure the proxy
- Restart the services and test the new SSL enabled Cloud

```
# apt-get install lighttpd
# apt-get install ssl-cert


# /usr/sbin/make-ssl-cert generate-default-snakeoil
# cat /etc/ssl/private/ssl-cert-snakeoil.key /etc/ssl/certs/ssl-cert-
snakeoil.pem > /etc/lighttpd/server.pem
```

Configuring the Public Cloud

SSL Security to access the EC2 Server

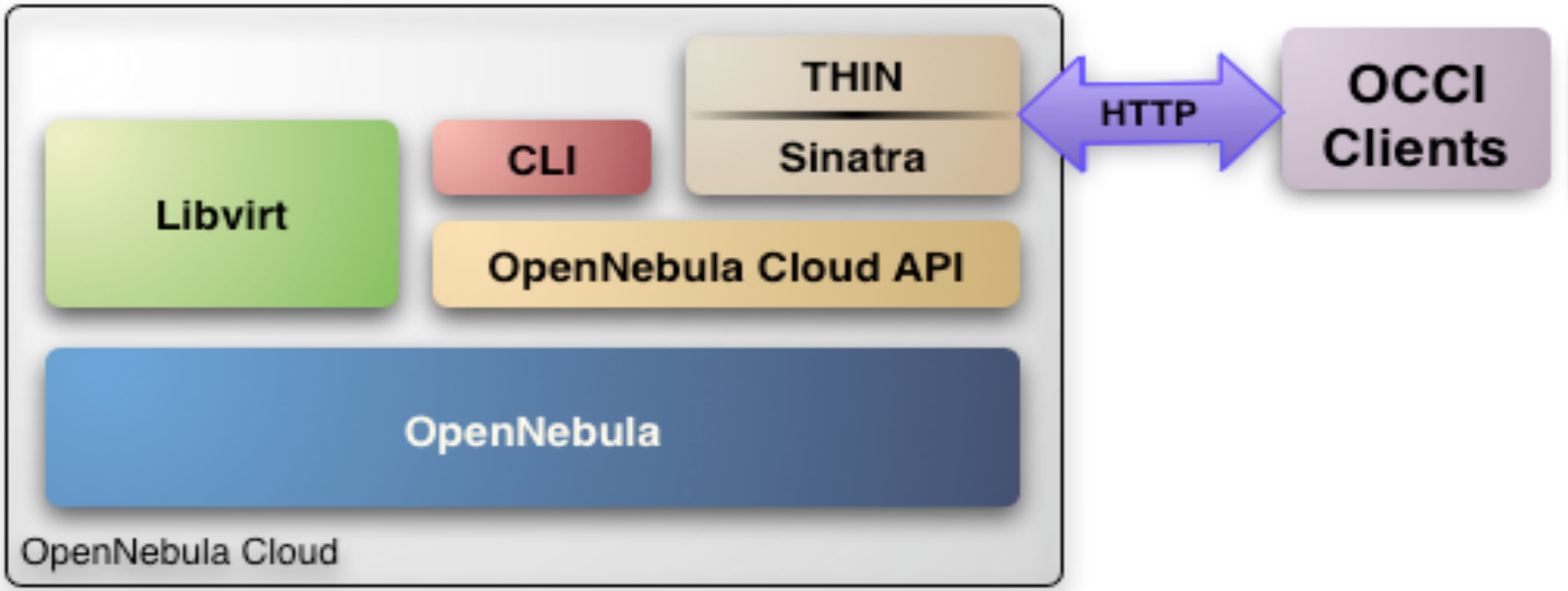
```
# vim /etc/lighttpd/lighttpd.conf
server.modules                = (
...
            "mod_compress",
            "mod_proxy"
...
## bind to port (default: 80)
server.port                   = 443
...
#### proxy module
proxy.server                  = ( "" =>
                                  ("" =>
                                   (
                                     "host" => "127.0.0.1",
                                     "port" => 4567
                                   )
                                   )
                                )
#### SSL engine
ssl.engine    = "enable"
ssl.pemfile   = "/etc/lighttpd/server.pem"
```

Configuring the Public Cloud

SSL Security to access the EC2 Server

```
$ vim /srv/cloud/one/etc/econe.conf

#SERVER=node-15.opennebula.org
SERVER = 127.0.0.1
PORT=4568

# SSL proxy that serves the API (set if is being used)
SSL_SERVER = pcaulaX.opennebula.org

$ econe-server stop
$ econe-server start

# service lighttpd restart

$ econe-describe-instances -K oneadmin -S onecloud -U https://
pcaula7.cesga.es:443
oneadmin      i-6              0                running
192.168.169.1   m1.small
```

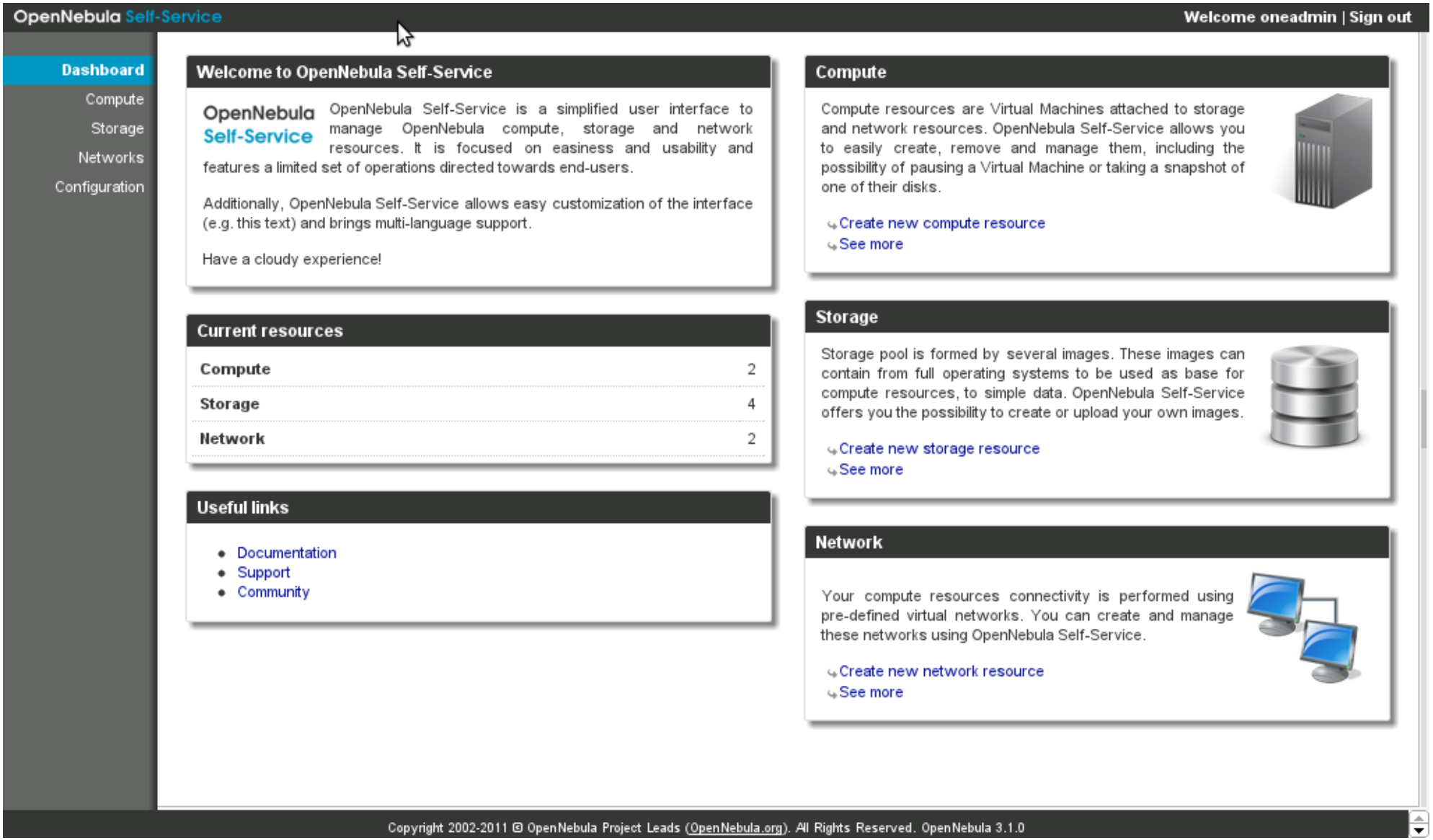
OCCI Overview



OCCI Overview

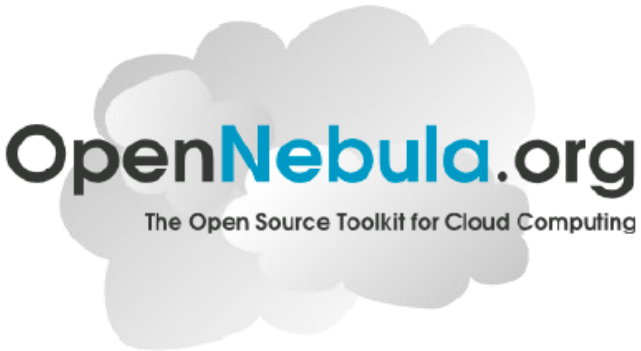
- Similar configuration as ECONE
- **REST protocol**
  - **Storage**
    - Upload: using a multi-part HTTP POST.
    - Retrieve: using a HTTP GET.
  - **Network**
    - Upload: using a HTTP POST.
    - Retrieve: using a HTTP GET.
  - **Compute**
    - Upload: using a HTTP POST.
    - Update: using a HTTP PUT.
    - Retrieve: using a HTTP GET.

OpenNebula Self-service



Building Clouds with OpenNebula 3.4
Public Cloud Computing

Constantino Vázquez Blanco
dsa-research.org | OpenNebula.org
Distributed Systems Architecture Research Group
Universidad Complutense de Madrid



- Public Cloud Computing with OpenNebula
- Installing a Public Cloud with EC2 API
- Configuring the Public Cloud
- Using the Public Cloud (EC2)
- Overview and Hands-On OCCI
- OpenNebula Self-service