Technische Universität München

Cloud Computing Course

WS - 2017

Exercise - 1

“Cloud Access & Security”

Authors:
Prof. M. Gerndt,
Prof. S. Benedict,
Anshul Jindal

1 Table of Contents

I. Introduction

1.1 Purpose of this document

This document provides guidance and material to assist the relevant person, in understanding the basic cloud infrastructure and security. This includes following:

- Creation of an account on a cloud provider.
- Creating a VM instance with a preconfigured system Image.
- Configuration of the VM
  - Public/Private IP allocation
  - Storage Device settings
  - Security Protocols
  - Creation of private and public key pair
  - Adding/Removing of user
  - User access authentication
- Creation of basic client Node.js application that sends VM information to a server machine on cloud.
- Access restriction to certain users
  - VM level restriction
  - Group level restriction
- Service Level Authentication
- Deployment of application on the cloud.

1.2 Prerequisites

Following requirements are mandatory:

- Account on LRZ

Following requirements are recommended (not mandatory):

- Basic Information about the organization of web infrastructure, protocols, servers, etc.
- Basic knowledge of Node.js as well as basic Linux console commands.

1.3 Background

1.3.1 Cloud Computing Platforms

1.3.1.1 Amazon AWS

Amazon Web Services (AWS), a subsidiary of Amazon.com, offers a suite of cloud-computing services that make up an on-demand computing platform. These services operate from 16 geographical regions across the world. They include Amazon Elastic Compute Cloud, also known as "EC2", and Amazon Simple Storage Service, also known as "S3". AWS has more than 70 services, spanning a wide range, including compute, storage, networking, database, analytics, application services, deployment, management, mobile, developer tools and tools for the Internet of things.

1.3.1.2 OpenNebula

OpenNebula is a cloud computing platform for managing heterogeneous distributed data center infrastructures. The OpenNebula platform manages a data center's virtual infrastructure to build private, public and hybrid implementations of infrastructure as a service. OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services (e.g. compute clusters) as virtual machines on

distributed infrastructures, combining both data center resources and remote cloud resources, per allocation policies.

1.3.1.3 OpenStack

OpenStack is a free and open-source software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS). The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center. Users either manage it through a web-based dashboard, through command-line tools, or through a RESTful API. OpenStack has a modular architecture with various code names for its components:

- **Keystone (Identity Service)**
  It's the main authentication and authorization service. It authorizes Users, Services, and Endpoints. Keystone uses tokens for authorization and maintains Session state.

- **Nova (Compute)**
  Nova compute or the king service provides a platform on which we are going to run our guest machines; It's the virtual machine provisioning and management module that defines drivers that interact with underlying virtualization. It provides a Control plane for an underlying hypervisor. Each hypervisor requires a separate Nova Instance. Nova supports almost all hypervisors.

- **Glance (Image Service)**
  In simple words glance is the Image Registry, it stores and Manage our guest (VM) images, Disk Images, snap shots etc. It also contains prebuilt VM templates so that you can try it on the fly. Instances are booted from glance image registry. User can create custom images and upload them to Glance for later reuse. A feature of Glance is to store images remotely so as to save local disk space.

- **Swift (Object Storage)**
  Swift Offers cloud storage software, look at it as Dropbox or Google drive, as they are not attached to servers, they are individual addressable objects. It's built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound. Swift provides redundancy checksum for files.

- **Cinder (Block Storage)**
  Cinder is also one of the storage modules of OpenStack; Think of it as an external hard drive or like a USB device. It has the performance characteristics of a Hard drive but much slower then Swift and has low latency. Block Volume are created in swift and attached to running Volumes for which you want to attach an extra partition or for copying data to it. It survives the termination of an Instance. It is used to keep persistence storage. Cinder Images are mostly stored on our shared storage environment for readily availability. These Images can be clones and snapshot which can be turned in to bootable images. It's like Amazon Elastic Block Storage.

- **Neutron (Networking)**
  Neutron, the networking component which was formally called Quantum. This component provides the software defined Networking Stack for OpenStack. It Provides networking as a service. It gives the cloud tenants an API to build rich networking topologies, and configure advanced network policies in the cloud. It enables innovation plugins (open and closed source) that introduce advanced network capabilities which let anyone build advanced network services (open and closed source) that plug into OpenStack tenant networks means that you can create advance managed switches and routers. You can even create an intelligent switch from a PC (Yes you can use it as a standalone component) and use it to replace your managed switch or at least make it act as a backup Switch.

- **Heat (Orchestration)**
  It creates a human and machine-accessible service for managing the entire lifecycle of infrastructure and applications within OpenStack clouds. It contains human readable templates with simple instruction that is read by the Heat Engine. Heat along with ceilometer (explained below) can create an auto-scaling feature of the cloud.

- **Telemetry (Ceilometer)**
  This module is responsible for metering Information. It can be used to generate bills based on the statistics of usage. Its API can be used with external billing systems. Administrators can create certain alarms that are triggered based on performance statistics

- **Horizon (Dashboard)**
  Horizon is the Dashboard to OpenStack, your eyes, and ears. It provides a web based user interface to OpenStack services including Nova, Swift, Keystone etc.

1.3.2 Cloud Security

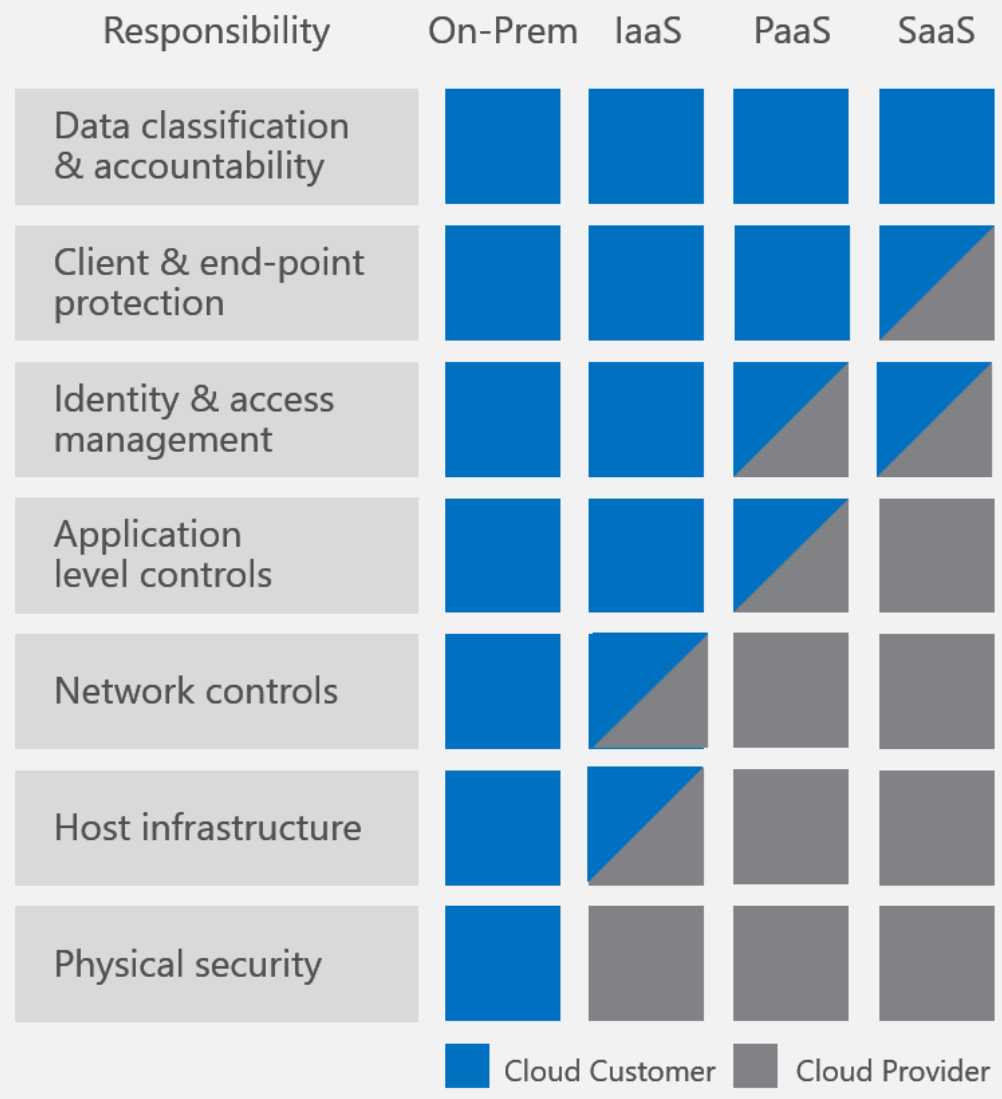
Cloud security refers to a broad set of policies, technologies, and controls deployed to protect data, applications, and the associated infrastructure of cloud computing.

With on-premises infrastructure, you have complete visibility and control over everything. You can physically see your infrastructure, and if something goes wrong, you have the power and ability to take immediate actions to fix issues.

But with cloud services, you need to trust your provider to properly secure your environment and respond to any security incidents in a timely manner, and as the data shows, many IT departments are hesitant to relinquish control and are afraid of outages or data loss that might occur in the cloud. Cloud computing and storage provides users with capabilities to store and process their data in third-party data centers. Organizations use the cloud in a variety of different service models (with acronyms such as SaaS, PaaS, and IaaS) and deployment models (private, public, hybrid, and community). Security concerns associated with cloud computing fall into two broad categories:

- security issues faced by cloud providers (organizations providing software-, platform-, or infrastructure-as-a-service via the cloud)
- security issues faced by their customers (companies or organizations who host applications or store data on the cloud).

The responsibility is shared, however. The provider must ensure that their infrastructure is secure and that their clients' data and applications are protected, while the user must take measures to fortify their application and use strong passwords and authentication measures.



In above Figure, the left-most column shows seven responsibilities that organizations consider, all of which contribute to the security and privacy of a computing environment.

Data classification & accountability and Client & end-point protection are the responsibilities that are solely in the domain of customers, and Physical, Host, and Network responsibilities are in the domain of cloud service providers in the PaaS and SaaS models. The remaining responsibilities are shared between customers and cloud service providers. Some

responsibilities require the CSP and customer to manage and administer the responsibility together, including auditing of their domains.

Authentication is the process of determining the identity of a client, which is typically a user account or a service account. Authorization is the process of determining what permissions an authenticated identity has on a set of resources.

1.3.3 Introduction to Node.js

Node.js is a JavaScript runtime that transforms JavaScript into one of the wide-purpose languages. Node.js applications can easily be used to process the data on the distributed systems according to some algorithm or set of algorithms. Node.js is supported by cloud computing systems. To effectively use Node.js for your Cloud applications, in this section we provide basic instructions on how to use Node.js for development of applications. Node.js exploits event-driven approach, so the development of Node.js application is mostly about writing processing functions for specific kinds of events.

1.3.3.1 Application Architecture

Node.js application consists of the following components:

- Import required modules – We use the require directive to load Node.js modules.
- Create server – A server which will listen to client's requests like Apache HTTP Server.
- Read request and return response – The server created in an earlier step will read the

HTTP request made by the client which can be a browser or a console and return the response. These components support the application architecture based on receiving the request, processing it and returning the response, it is just a simple client-server interaction.

1.3.3.2 Import required modules

To import all required by your application modules, you should use require directive. The use of this directive is as follows:

```
let http = require("http" );
```

By using this statement in your application, you create HTTP instance and store it in the http variable. Now you can access methods of this instance to provide your application with HTTP server.

1.3.3.3 Create Server

You can create http server using HTTP instance stored in http variable. The most important thing that you need to do is to create the server instance and bind it at a port (e.g. 8081). HTTP server instances are created using the call to a http.createServer() method, whereas binding server at a port is fulfilled by using listen(port) method. The following code chunk provides an example of a simple HTTP server, which emits the same response for each incoming request. Note that we give to createServer method a function as an argument. This function describes the way in which the server will process the incoming requests and provide responses.

```
http.createServer( function( request, response )
{
```

```
        // Creating a header for a response
        response.writeHead( 200, { 'Content-Type': 'text/plain' });
        // Creating response body to send
        response.end("Test response" );
}). listen (8081);
// Printing a message in a console
console.log ('Server running at http://127.0.0.1:8081/');
```

To test it, just transfer both sections to a single file named test. Then run the following command in the console:

node test.js

You should see a string in the console window: Server running at http://127.0.0.1:8081/. Now try and open http://127.0.0.1:8081/ in your browser. You should be able to see the text "Test response"

1.3.3.4 Blocking and Non-Blocking Code

Node.js introduces a concept of callbacks – asynchronous equivalents of functions. The main advantage of callbacks is their ability to return the control to next parts of program thus providing program with the ability to continue execution while the callback function processes something. Callbacks usage helps you to use your server's resources more wisely. It is important to note that callback functions are called when asynchronous function returns its results.

1.3.3.5 Event Loop

Node.js heavily relies on the concept of event-driven programming. In event-driven application there usually is a main loop listening for events. This loop triggers a corresponding callback function when it detects one of the events. The listening functions act as observer: when a corresponding event gets fired, its listener function starts executing. In order to use events as a part of your application, you need to import events module:

```
let events = require ('events' );
```

1.3.3.6 Express Framework

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Express allows to set up middleware to respond to HTTP requests, also it allows dynamical rendering of HTML pages and defines a routing table to perform different actions based on HTTP method and URL. In order to install express, use the following command:

npm install express --save

Here is an example of application that starts a server and listens on port 3000 for connection. This application responds with "Hello World!" for requests to the home page, whereas for every other path it responds with "404 Not Found":

```
var express = require('express');
var app = express ();
app.get('/', function (req, res)
{
```

```
        res.send('Hello World');
});
let server = app.listen(8081, function ()
{
        let host = server.address().address
        let port = server.address().port
        console.log ("Listening at http://%s:%s", host, port)
})
```

Save the above code in a file named server.js and run it as usual. You should see the following output: ‘‘Example app listening at http://0.0.0.0:8081’’. In order to see a page with the response, just open in the browser the following link http://127.0.0.1:8081/

Express application uses a callback function whose parameters are request (req) and response (res) objects:

- Request Object – The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
- Response Object – The response object represents the HTTP response that an Express app sends when it gets an HTTP request.

II. Exercise Steps

2.1 Cloud Account Creation and Access

2.1.1 OpenNebula

- For accessing the cloud, first step is to get your username and password.
- After you have received username and password, you can access the cloud. The Cloud management GUI is available at https://www.cloud.mwn.de. Log in with your LRZ username and password.

Figure 1: OpenNebula Login Page

- After a successful login, you will be presented with a dashboard (detailing the resources used by you) and the main menu, on the left part of the web interface, where it is possible to familiarize yourself with the main concepts of ONE (web based Graphical User Interface (GUI), also known as Sunstone*)*.

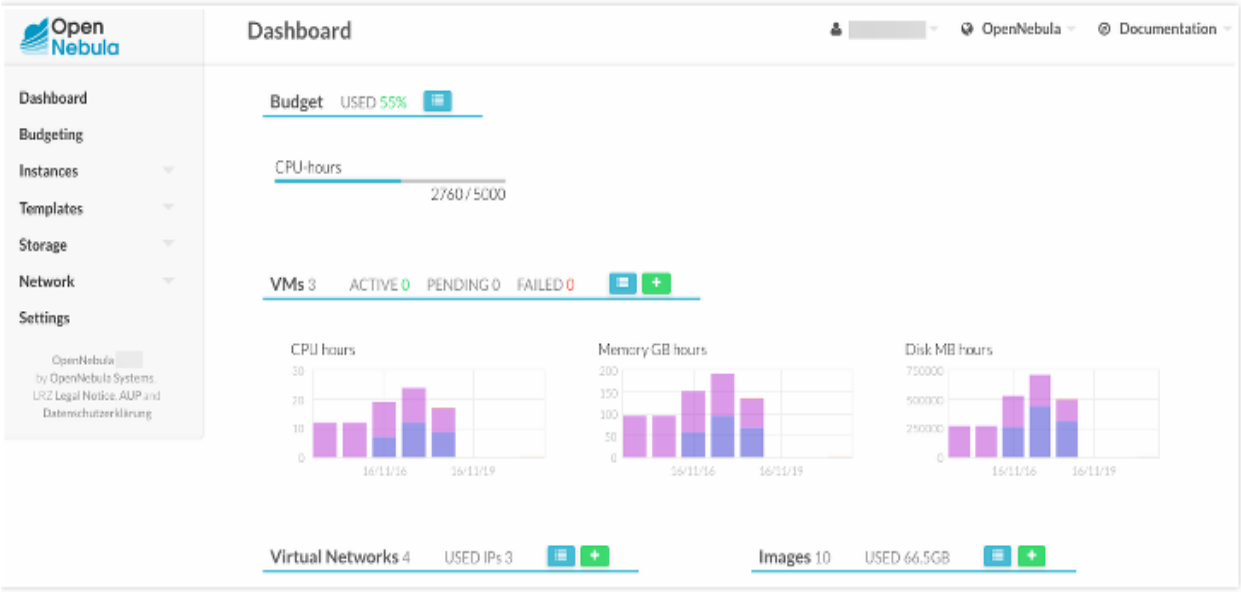


Figure 2: OpenNebula Dashboard

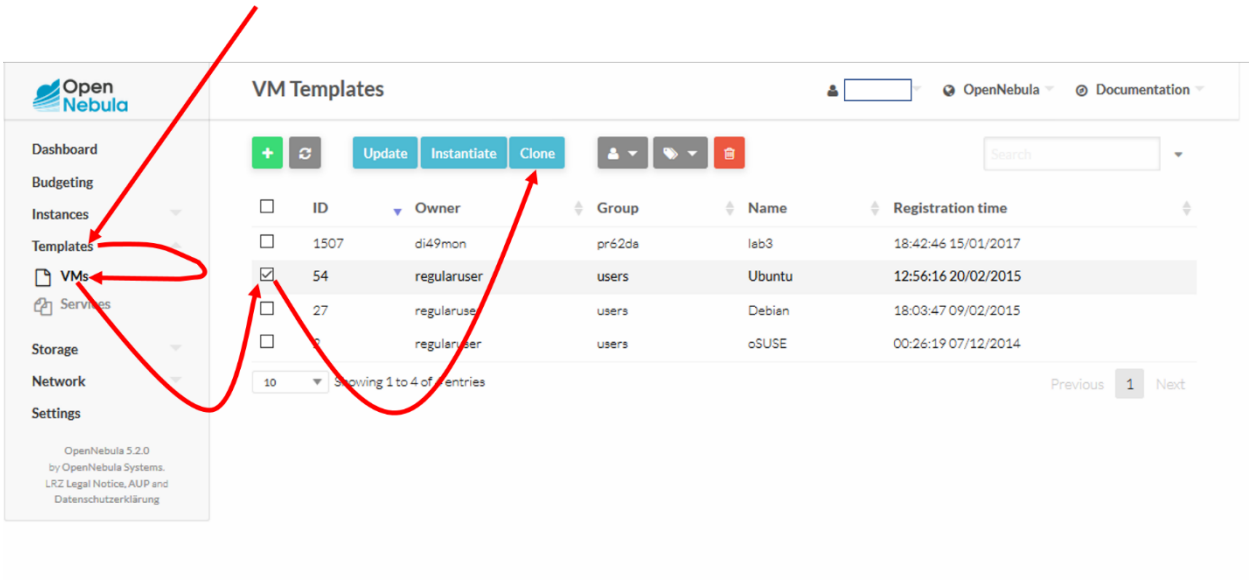
- Details of the various options on the left menu:
    - **Budgeting:** Budget accounting. Data collection runs at one minute past the hour and records the consumption in the last time window (last hour). This means that a fraction of an hour can be accounted and the system is thus more fair towards the users. If a VM is launched, for instance, twenty minutes after the script has run, at the next sampling time only the forty minutes of effective usage will be deducted. In fact, the granularity of the accounting system is one second.
    - **Instances of Virtual Machines:** This view shows the user a summary of the running VMs, i.e., the instantiated templates, including all the details, such as the allocated IP(s). From here, it is also possible to open a VNC window to have direct access to a particular instance.
    - **Instances of running Services:** The list of multi-tier applications running in OneFlow(A framework to build and manage elastic services, made up of many VMs, organized on different layers).
    - **Templates of Virtual Machines:** This is where the user shapes the resource he/she wants to allocate.
    - **Templates of Services:** The list of multi-tier application defined to work by means of OneFlow.
    - **Datastores:** It's the physical space to host the images. Usually the user does not have control of it.
    - **Images***:* The disk images for the VMs. An image can contain the operating system (OS) to boot a VM, a CDROM image to install a VM or it can be a datablock device to have spare space for data. An image is one of the items that the user should

customize (with his/her OS, user environment, and applications) and this capability is the rationale behind the ONE service. However, please keep in mind that only raw and qcow2 formats are supported.

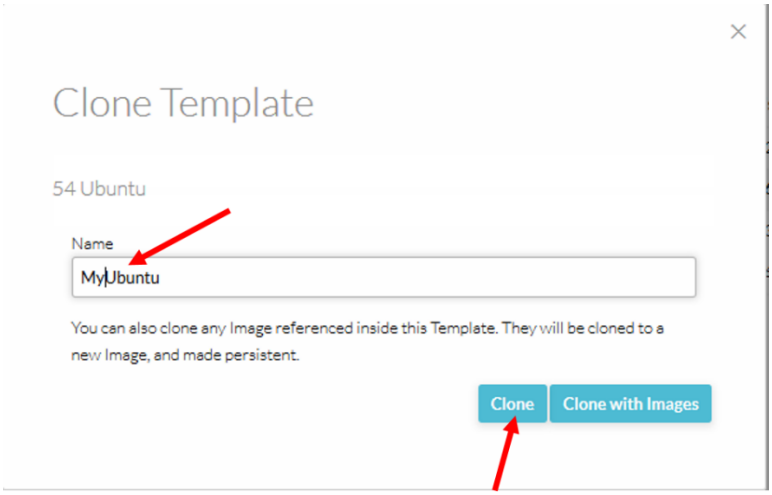
- **Files***:* This section is used in some advanced cases, when the user wants to boot a machine using a kernel not contained on the disk image (currently discouraged by LRZ) or for customization purposes.
- **Virtual Networks:** The networks that can be accessed by the running VMs. Usually the LRZ provides access to a pool of public routable IPs and to the internal LRZ network (with a gateway for outgoing traffic to the general Internet), including the usual LRZ services such as NAS, TSM backup, and the MWN network.
- **Settings:** A panel with some additional information. It is used to set up the default SSH public key to be injected in the virtual machines or the password for the OpenNebula's EC2 interface and other stuff.

2.2 Creation, Configuring and Starting a VM

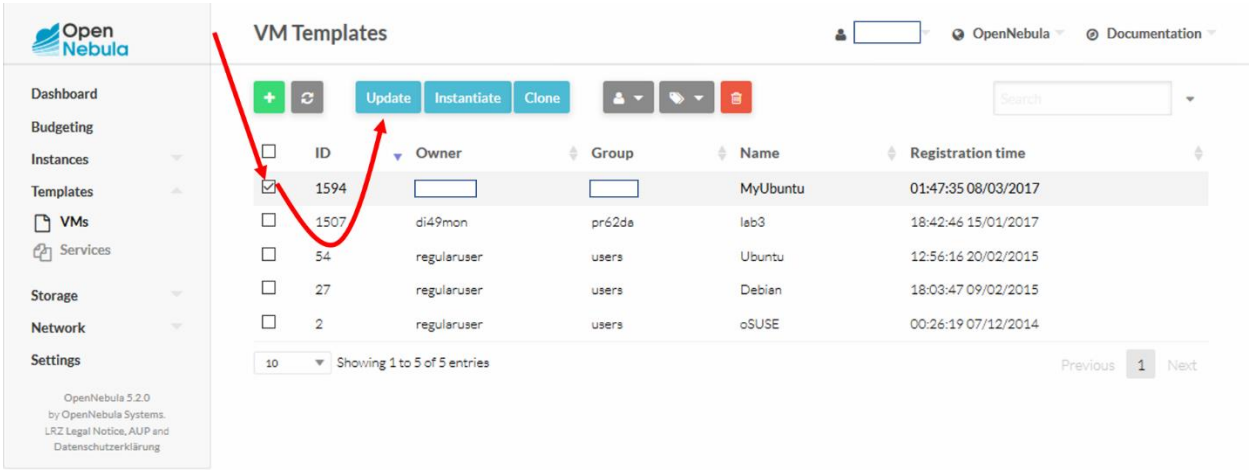
- After a successful login, you will be presented with a dashboard.
- Now we select a template for our Ubuntu VM. Luckily, this has already been prepared, so all we have to do is select it. Click on Templates ➔ VMs and then checkmark the Ubuntu template, as shown in below figure



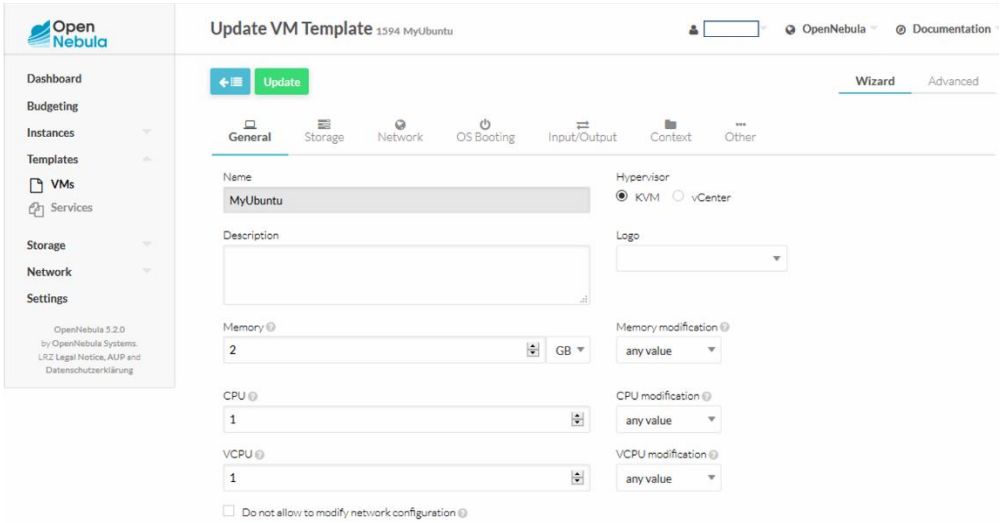
- Now we have to modify the template a bit, so that we will be able to log into our new VM after we start it. To do this, we need to make a private copy of the template, so we click "Clone". We name this template and clone it by pressing "Clone" in the pop-up window.



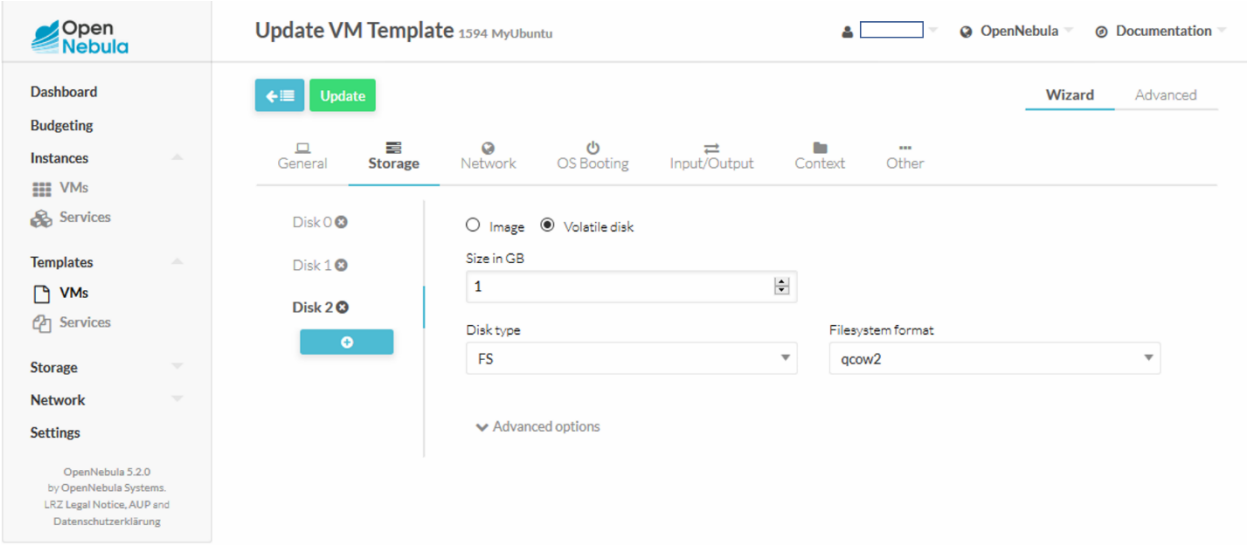
- Now we select this private template, which we can modify in the next step and press "Update"



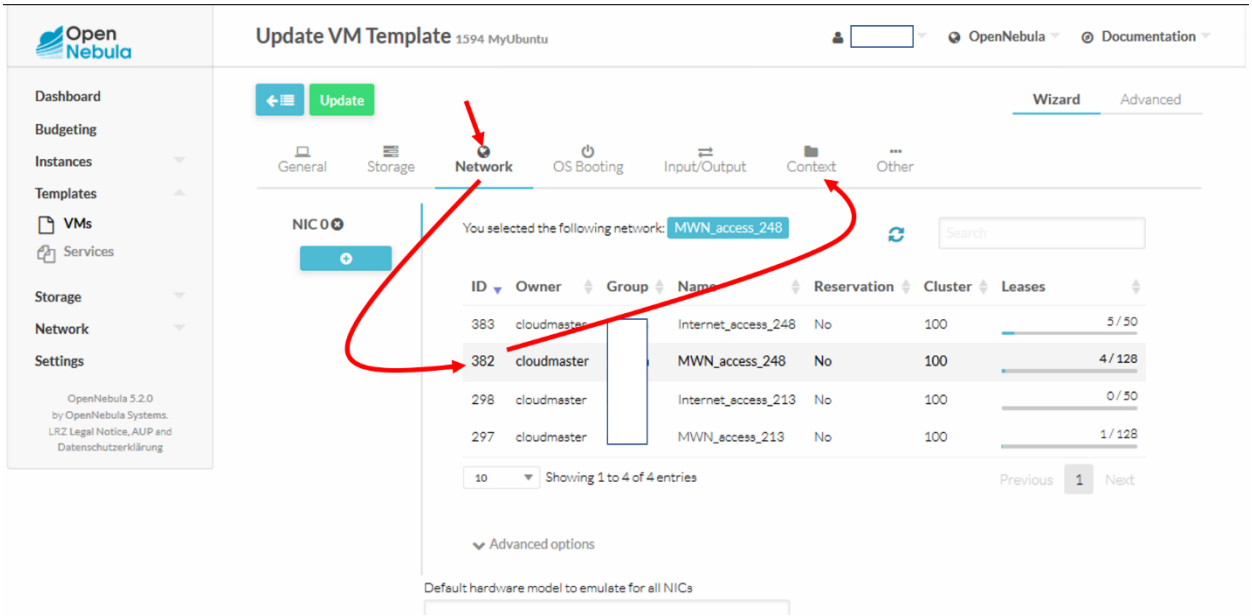
- After clicking update, you will get the update wizard.



- Click on the storage tab, here you can attach a new Disk (image disk or Volatile disk) or increase the size of already attached disk. For this exercise we will leave everything to default.

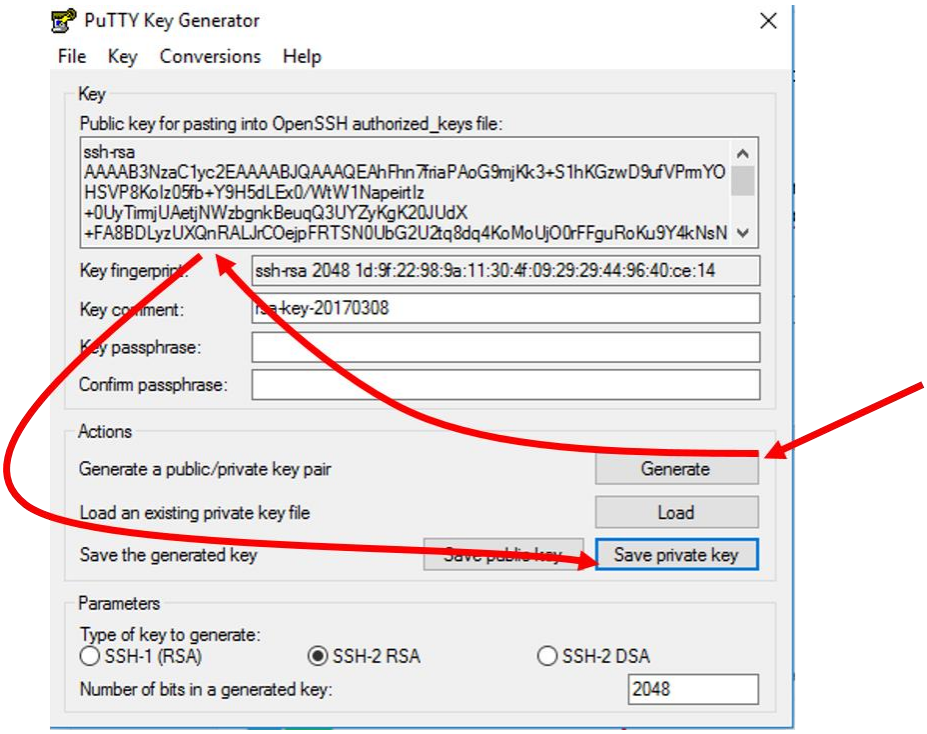


- Now, we want to add a network interface to our VM so that we will be able to log in from the outside. But be cautious and only select an interface to the MWN.

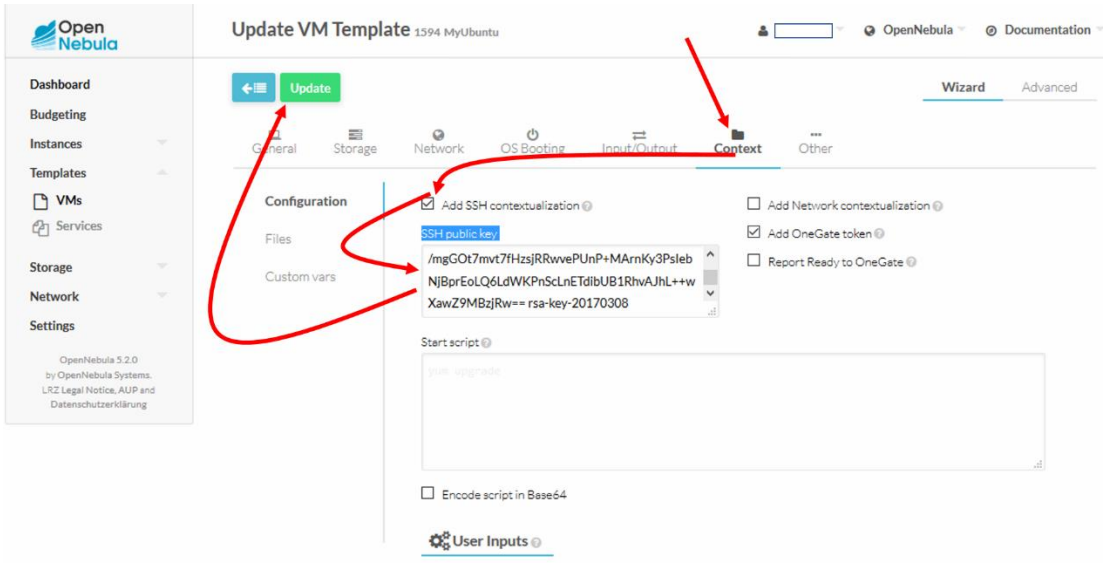


- We want to insert our ssh key via the contextualization so that we don't need a password to log into the root account.
  - Click on the "Context" item from the ribbon.
  - Check the "Add SSH contextualization" box.

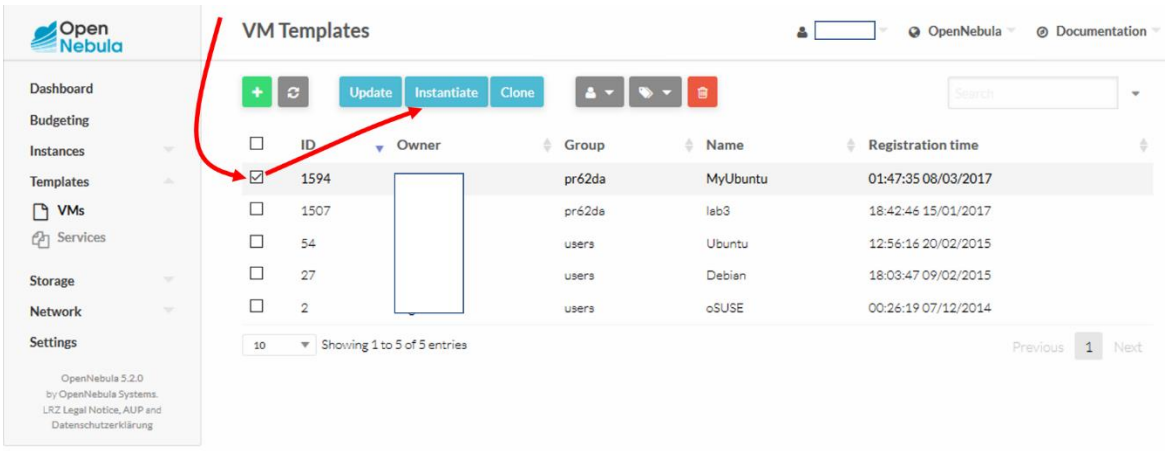
- Copy your ssh key into the text field. In Linux generate the key pair using the following command **"ssh-keygen -t rsa"** or for windows you can use puttygen to generate public key. Also save the private key to use it with putty.



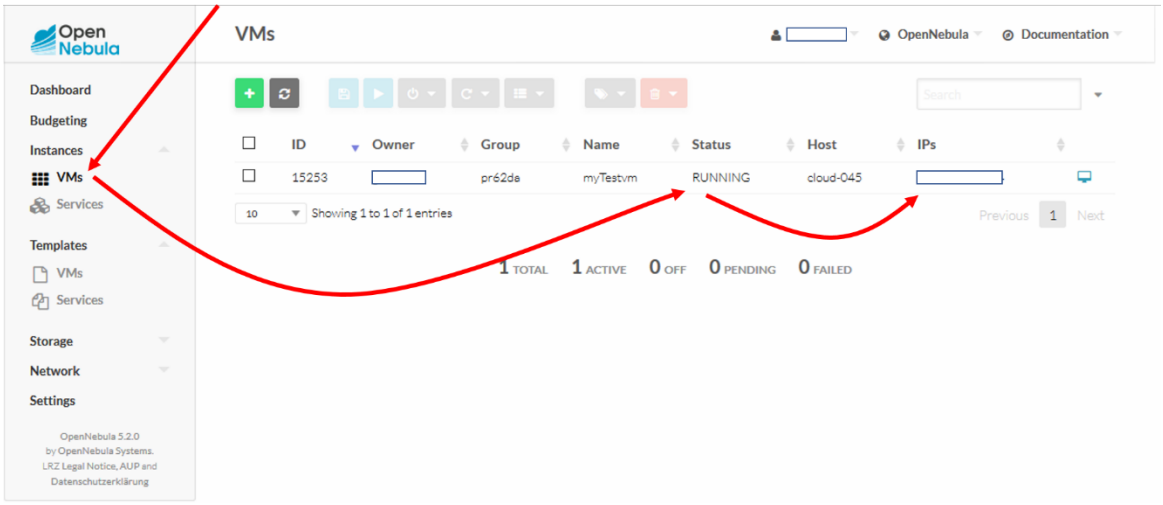
- Then click on "Update". In the below figure Start script box is used when you want to run a particular script when the VM starts.



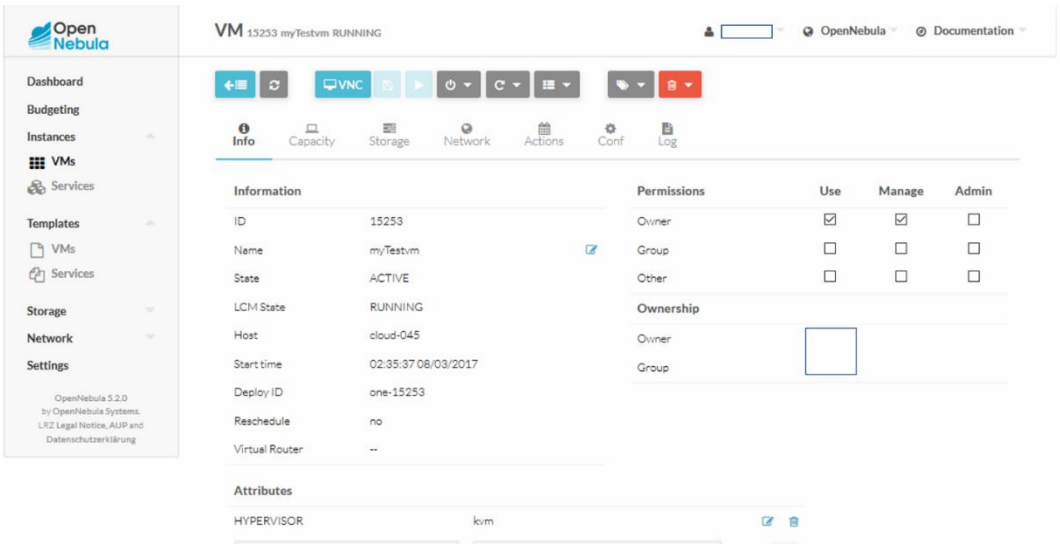
- Now you can instantiate (activate) your VM. Click the "Instantiate" button:



- Name your VM, so you can identify it later on. Here we call it "myTestvm". Then Click "Instantiate".
- After a while you can see your new VM in the "Virtual Machines" display: 1 active VM! You can even see it's IP address also. You may have to click on the refresh button (don't reload the page!) several times until you see the VM and then a bit later its IP.



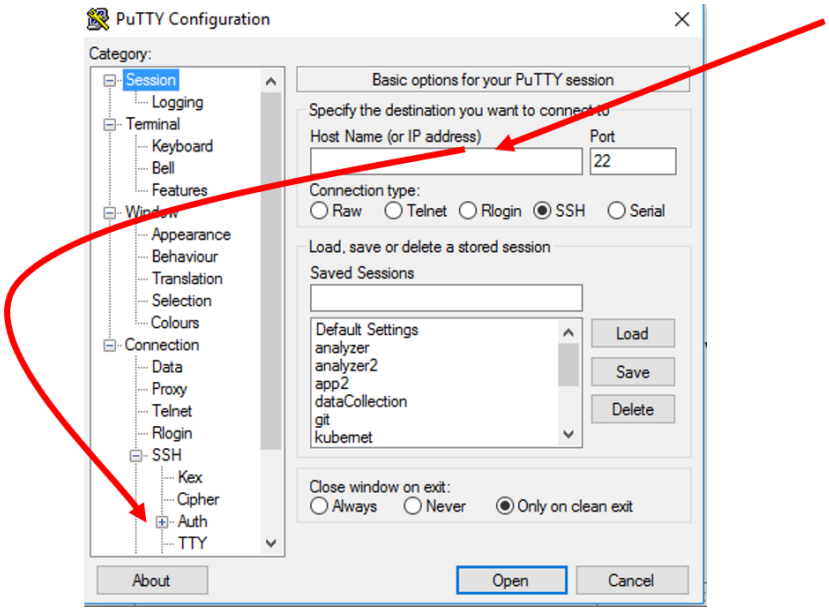
- By clicking on the VM you can get to know the detail information of the VM.

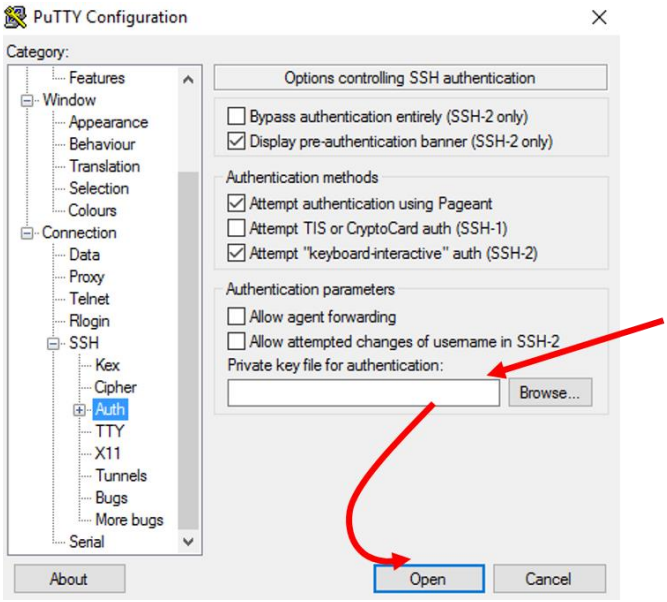


- You can now ssh from your local computer to your VM

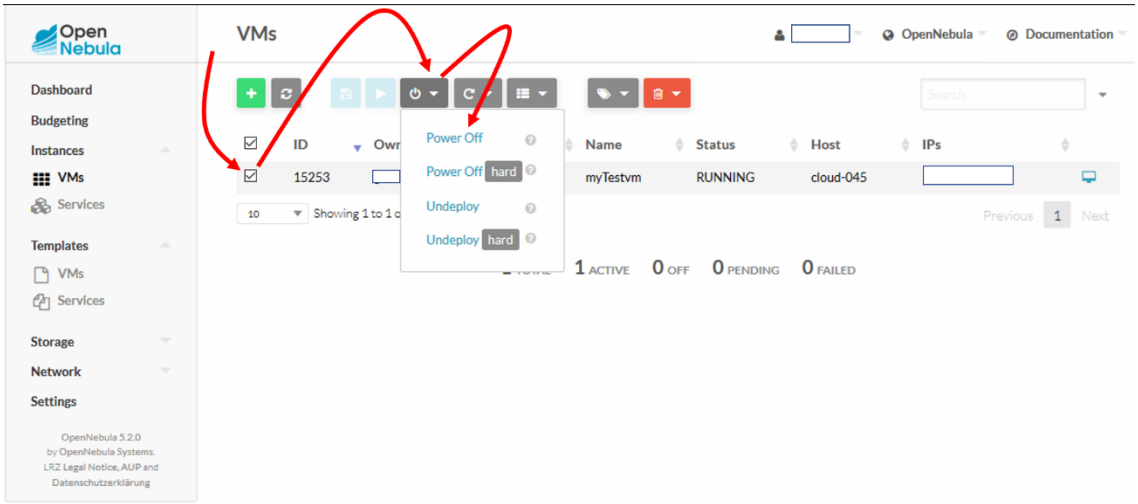
  ssh -i Key_Path root@IP_Address

  Or you can use putty if you are using windows.





- Once you are logged in as root, you should change the root password by typing the command "passwd".
- Now you have learned to start the VM, enjoy your new VM and undeploy it down once you no longer need it, to save resources! (But don't power off now we need to run further steps)



Note:

Power off: gracefully power off the machine via ACPI, preserving its disk(s) but not its state (i.e., the RAM). When resumed, the VM is booted from scratch, on the same node where it was before. It is used to save boot disks and to take	***Undeploy****:* gracefully power off the machine via ACPI, preserving its disk(s) but not its state (i.e., the RAM), freeing the worker node, i.e., CPU(s) and RAM return available for other users. When resumed, the VM will boot from scratch selecting the first available host. The *hard* version of the command does not use ACPI, hence

disk snapshots. The *hard* version of this command does not use ACPI.	the guest OS will be abruptly terminated, like pulling the power plug.

2.3 VM / Cloud Level Authentication

2.3.1 User Level Authentication

Now the task involves how to add the users and give access to the VM

Please follow the following steps:

1. Login into your VM as stated in the previous step.
2. Now you can see your terminal, so the first step involves adding the user. Type the following command to add the user

   useradd -m -d /home/username -s /bin/bash username

   Where username is any username you want to add.

   -s, --shell SHELL
   The name of the user's login shell. The default is to leave this field blank, which causes the system to select the default login shell specified by the SHELL variable in /etc/default/useradd, or an empty string by default.

   -m, --create-home
   Create the user's home directory if it does not exist. The files and directories contained in the skeleton directory (which can be defined with the -k option) will be copied to the home directory. By default, if this option is not specified and CREATE_HOME is not enabled, no home directories are created

   -d, --home HOME_DIR
   The new user will be created using HOME_DIR as the value for the user's login directory. The default is to append the LOGIN name to BASE_DIR and use that as the login directory name. The directory HOME_DIR does not have to exist but will not be created if it is missing.

3. Now make the directory with the name ".ssh" inside the new user home directory. This directory will store our ssh keys. Use the following command to make directory

   mkdir /home/username/.ssh

4. Now copy your public key to the new user's authorized_keys with the following command:

   cp /root/.ssh/authorized_keys /home/username/.ssh/authorized_keys

   Note: You can even create a new key and then copy that public key to the user's authorized keys but for this exercise we will be using the already added public key so that you can use your earlier private key to log in with new user.

5. Now set correct permissions to files on new user with the following commands

   chown -R username:username /home/username/.ssh
   chmod 700 /home/username/.ssh

chmod 600 /home/username/.ssh/authorized_keys

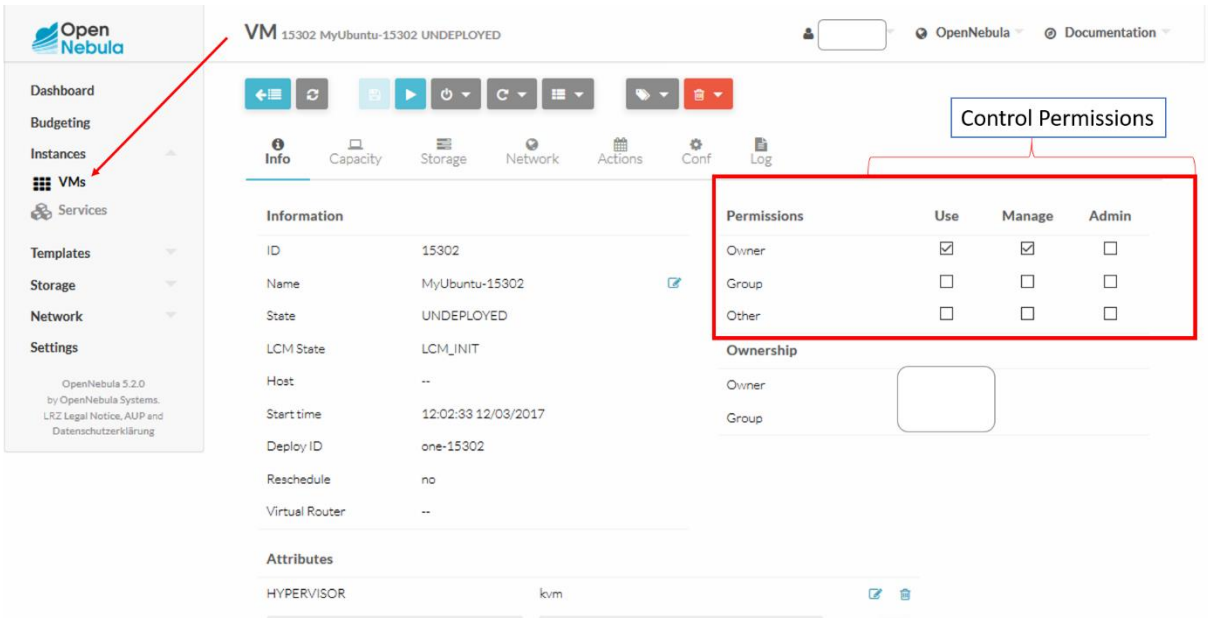
6. Now the new user is ready. Open another terminal or putty instance and login with this new user name and your earlier private key.

   Note:

   As part of exercise, create at least one more user and login with that.

2.3.2 Group Level Authentication

In the OpenNebula your account is created under a group, which has certain number of users. If you want to share your VM to a group, then you can enable it from the dashboard. As shown in the below figure:



Here on the right side, under the permissions you can manage to whom you want to share your VM with.

Apart from group you can even provide access to users who are not part of your group by enabling the **other.**

2.4 Service Level Authentication

Here you are going to learn how to implement an authentication at service level using Node.js. As part of this exercise we will be ldoing basic authentication. A more flexible and robust can be implemented by using different node.js libraries like Passport.js.

2.4.1 Global Authentication

The first technique is authentication globally to every route. If you want to secure every service, then add the authentication globally.

Here we are using basic authentication so there would be no database to store user name and password but for advanced application development a database can be attached and queries for username and password.

2.4.2 Single-Route Authentication

This technique is used If you want to secure some services and allow other services without any access. Just like before we define our authentication handler, except this time we need to define it inside the service for which we want authentication.

2.5 Node.js Client Application Development

Now the next task is to make a simple Node.js application. This application will be used to get the information about the VM on which it is deployed. Follow the following steps:

2.5.1 Install the node and npm

Ubuntu

1. Get yourself a more recent version of Node.js by adding a PPA (personal package archive) maintained by NodeSource. This will probably have more up-to-date versions of Node.js than the official Ubuntu repositories.

   curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
2. You can now install the Node.js package.

   sudo apt-get install nodejs
3. The nodejs package contains the nodejs binary as well as npm, so you don't need to install npm separately. However, for some npm packages to work (such as those that require building from source), you will need to install the build-essentials package

   sudo apt-get install build-essential

Windows

1. Download the Windows installer from the Nodes.js web site.
2. Run the installer (the .msi file you downloaded in the previous step.)
3. Follow the prompts in the installer (Accept the license agreement, click the NEXT button a bunch of times and accept the default installation settings).
4. Restart your computer. You won't be able to run Node.js until you restart your computer.

2.5.2 Test the Installation of the node and npm

Make sure you have Node and NPM installed by running simple commands to see what version of each is installed

Test Node:

To see if Node is installed, open the Terminal / Windows Command Prompt / PowerShell or a similar command line tool, and type

node -v.

This should print a version number, so you'll see something like this v0.10.35.

Test NPM:

To see if NPM is installed, type

npm -v

This should print NPM's version number so you'll see something like this 1.4.28

2.5.3 Download the provided clientApplication

Download the client application provided as part of this exercise. It is the basic application which consist of creating an express server and providing routes for the API.

2.5.4 Installation of required modules.

As part of this application some modules need to be installed. For installing them run the following command from inside the directory of application.

npm install

This command will install all the dependent modules mentioned in the package.json file.

Note:

If you need some other modules you can install them by running the command

npm install "module name" –save

This will automatically add that module in the package.json file and now you can use it inside you development file.

2.5.5 Explanation of clientApplication

A brief explanation of different part of the application are here:

Base Setup

This section consists of importing the required packages. For detail about the packages you can check on Nodes.js web site. Some brief comments are added to explain other parts.

```
// ===================================================================================
/**
 * Cloud Computing Cource Exercises
 * Exercise 1
 *  2 Tasks
 *      1. Accessing VM information using unauthenticated API
 *      2. Service Level Authentication
 * Developed by 'Write Group Name'
 * Write Names of All Members
 */
// ===================================================================================
/**
 * BASE SETUP
 * import the packages we need
  */
const express    = require('express');
const app        = express();
const port       = process.env.PORT || 8080; // set our port
/**
```

Routes

This section describes the routes information. Basically, this section covers the part when someone visits your application URL, then what all commands you want to run and what all information you want to return to the user.

```
/**
 * ROUTES FOR OUR API
 * Create our router
 */
const router = express.Router();
/**
 * Middleware to use for all requests
 */
router.use(function(req, res, next) {
    /**
     * Logs can be printed here while accessing any routes
     */
    console.log('Accessing Exercises Routes');
    next();
});
/**
 * Base route of the router : to make sure everything is working check http://localhost:8080/exercises)
 */
router.get('/', function(req, res) {
    res.json({ message: 'Welcome to Cloud Computing Exercises API!'});
});
```

Exercise Message:

This section describes the structure of exercise message which will be filled in the exercise1_task1 route and sent to the server.

```
// ====================================================================================================================
/**
 * TO DO
 * 1. Get the number of current users login into virtual machine
 * 2. Get the names of those users
 * 3. Get the number of storage disks ((we are here only concerned about the disks and that too Virtual disks (vd)))
 * 4. Get size Information about the above disks (disk: size).
 * 5. save in exercise_1_Message
 */
// ====================================================================================================================
let exercise_1_Message = {
        message: 'exercise_1',
        numberUsers: 'x',
        userNames:['x','y'],
        numStorageDisks:'xy',
        storageDisksInfo:['size1', 'size2', 'size3']
    };
```

Server Start:

This section starts the server at the given port.

```
// START THE SERVER
// =============================================================================
app.listen(port);
console.log('Server started and listening on port ' + port);
```

2.5.6 Tasks to be completed

As part of the exercise1, there are two subtasks to be completed **(in two separate APIs):**

1. **Access and send following VM information in unauthenticated API**
    1. Number of users logged in (Remember to add at least one more user and try login through it).
    2. Usernames of the user logged in. Usernames of the users are to be specified in the array as shown in the exercise message structure template.
    3. Number of storage disks attached (we are here only concerned about the disks and that too Virtual disks (vd))
    4. Information about the above disks (disk: size). Information of the disks are to be specified in the array as shown in the exercise1Message structure template.
2. **Add a authenticated API with default authentication to username "CCS" and password as "ccs_exercise1_task2". Here success and failure messages should also be added**

Hints:

1. You can use Linux shell commands to get all the above information.
2. For running the Linux shell commands inside node.js check the child_process module of node.js

2.6 Node.js Client Application Deployment

Now you have a VM running on the cloud and a node.js application. So next step involves the deployment of this node.js application. Follow the following steps:

1. ssh into the VM
2. Install the node and npm in the VM and test whether they are running or not.
3. If they are properly working, copy your node.js application to the VM Copy can be done in number of ways like using secure copy(scp) from your local machine to VM or by putting the code on GitHub and fetching the code in VM etc.
4. Now you have a VM running on the cloud with your node.js application code in it along with node and npm installed.
5. Next go inside the application directory, and install the application dependent module. You already know how to do this (By running npm install).
6. Now your application is ready to be deployed on VM. Run the following command to start the application:

   node "applicationName".js

   After running this, on console you will see

   "server started and listening on port 8080"
7. Now your application is deployed on the server. Open your browser on your local machine and enter the address as

http://IP_ADDRESS_VM:8080/exercises

Note:

If your request timed out, your VM probably has some firewall rules in place prevent a user to call your web server from the outside. The iptables rules are located in the file /etc/iptables/rules.v4. Open this file with your favorite editor:

```
# Generated by iptables-save v1.6.0 on Fri May  6 15:32:09 2016
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 4 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -m state --state NEW -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m limit --limit 3/min -j LOG
COMMIT
# Completed on Fri May  6 15:32:09 2016
~
```

After line 9 insert a new line allowing incoming connections on port 8080:

-A INPUT -p tcp -m tcp --dport 8080 -m state --state NEW -j ACCEPT

```
# Generated by iptables-save v1.6.0 on Fri May  6 15:32:09 2016
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 4 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -m state --state NEW -j ACCEPT
-A INPUT -p tcp -m tcp --dport 8080 -m state --state NEW -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m limit --limit 3/min -j LOG
COMMIT
# Completed on Fri May  6 15:32:09 2016
~
```

To apply the new iptables rules, you need to reload them to your local firewall system.

iptables-restore < /etc/iptables/rules.v4

Now you will be able to see your application running with following message:

'Welcome to Cloud Computing Exercises API

2.7 Result delivery

You already have done a lot of tasks, now it's the time to get grade for those tasks. To submit your application results you need to follow this :

1. Open the cloud Class server url
2. Login with your opennebula username and any password.
3. After logging in, you will find the button for exercise1
4. Click on it and a form will come up where you must provide your VM ip on which your application is running.

   Ex : 10.0.23.1

5. Then click submit.

Important points to Note:

1. Make sure your VM and your application is running after following all the steps mentioned in this manual.
2. We will grade you based upon the number of tasks completed by you.
3. You will get to see, what your application has submitted to the server.
4. You can submit as many times until the deadline of exercise.
5. Multiple submission will overwrite the previous results.

APPENDIX -1

(For further knowledge, not used as part of this exercise)

Passport.js

Passport is an authentication middleware for Node.js which is used for session management. Passport is authentication middleware for Node. It is designed to serve a singular purpose: authenticate requests. When writing modules, encapsulation is a virtue, so Passport delegates all other functionality to the application. This separation of concerns keeps code clean and maintainable, and makes Passport extremely easy to integrate into an application.

In modern web applications, authentication can take a variety of forms. Traditionally, users log in by providing a username and password. With the rise of social networking, single sign-on using an OAuth provider such as Facebook or Twitter has become a popular authentication method. Services that expose an API often require token-based credentials to protect access.

Passport recognizes that each application has unique authentication requirements. Authentication mechanisms, known as strategies, are packaged as individual modules. Applications can choose which strategies to employ, without creating unnecessary dependencies.

Redis

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

You can run atomic operations on these types, like appending to a string; incrementing the value in a hash; pushing an element to a list; computing set intersection, union, and difference; or getting the member with highest ranking in a sorted set.

To achieve its outstanding performance, Redis works with an in-memory dataset. Depending on your use case, you can persist it either by dumping the dataset to disk every occasionally, or by appending each command to a log. Persistence can be optionally disabled, if you just need a feature-rich, networked, in-memory cache.

REFERNCES

1. **LRZ Supercomputing Center**
   https://www.lrz.de/english/

2. **Node.js official website**
   https://nodejs.org/en/

3. **Node.js Tutorial**
   https://www.tutorialspoint.com/nodejs/

4. **Node Package Manager**
   https://www.npmjs.com/

5. **Representational state transfer protocol.**
   https://en.wikipedia.org/wiki/Representational_state_transfer

6. **JSON format.**
   https://en.wikipedia.org/wiki/JSON

7. **Express js framework.**
   http://expressjs.com/