

CSC3095:Web Platform for Digital Deployment of Virtual Servers

Plamen Kolev

Student number : 130221960

Supervisor : Neil Speirs

03.05.2017

1 Declaration

I declare that this dissertation represents my own work except where otherwise stated.

2 Acknowledgements

I would like to thank to Dr. Neil Speirs for the valuable support, feedback and resources he provided during the project, without which this work would be impossible. I would also like to thank Helen Griffiths and Chris Ritson from Newcastle University IT for finding time to help me with project evaluation. And finally, I want to thank to the open-source communities of Oracle, Ruby On Rails, Puppet and vendors like Intel for contributing with great examples, references and materials.

3 Abstract

This paper describes the process of building a platform for deployment of virtual servers. The methodology for creating the final product includes a research into virtualisation from leading vendors such as Oracle, Intel, Amazon and Microsoft. It covers creating the initial architecture, building and testing the core features and provides insights into the decisions made during the process. The document goes through the development, testing and evaluation process and describes the software patterns and practices of building such platform. These patterns include managing source-code with version control, having multiple development environments, writing unit tests and building software incrementally and using best practice guides from leading experts in the field.

Contents

1	Declaration	
2	Acknowledgements	
3	Abstract	
4	Introduction	1
4.1	Purpose	3
4.2	Demographic	4
4.3	Aim and Objectives	4
4.3.1	Aim	4
4.3.2	Objectives	4
5	Background	6
5.1	Virtualisation and "The Cloud"	6
5.2	Similar in the field	9
5.2.1	Amazon Web Services™EC2	9
5.2.2	Microsoft Azure	10
6	Work	12
6.1	Overview	12
6.2	Vagrant	13
6.3	Workflow	13
6.4	Languages and frameworks	14
6.5	The deeploy module - architecture	15
6.6	Virtual networks	17
6.7	Extending virtual machines	17
6.8	Building The Web Server	18
6.8.1	Database and authentication	19
6.8.2	Web Form	19
6.8.3	Asynchronous deployment	19
6.8.4	Implementing the Health monitor	20
6.8.5	Obtaining Certificates	21
6.8.6	Management Functionality	21
6.9	Security	21
6.9.1	Passwords and authentication	22
6.9.2	SSH Credentials without password	22
6.9.3	Guest Isolation concerns	23
6.9.4	SSH Keys	23
6.9.5	Source of packages	23
6.9.6	Development practices	23
7	Testing	25

8	Evaluation	26
8.1	Introduction	26
8.2	Deployment Overview	26
8.3	Deployment Time Results	26
8.4	Deployment Progress	28
8.5	Investigating Failures	28
8.6	Extensibility of solution	30
8.7	Health monitor	30
8.8	Web Interface	31
8.9	Authentication and Credential management	34
8.10	Documentation	34
8.11	Firewall and package management	35
8.12	Package Mirroring	35
8.13	System resource validation	35
	8.13.1 Security evaluation - Leaking passwords	35
	8.13.2 Allocation of network addresses	36
9	Conclusion	37
9.1	Fulfilment of requirements	37
9.2	Personal development	37
9.3	Future work	37
10	References	39
11	Glossary	41
	Appendices	44
A	Planning	44
A.1	First Semester	44
A.2	Second Semester	45
	A.2.1 Developmen	46
	A.2.2 Learning period	46
	A.2.3 Documentation	46

List of Figures

1	The two types of virtualisation [10]	1
2	VSphere data-store manager screen-shot. Source [18]	3
3	Heterogeneous server configuration. Source [24][p.59]	6
4	Amazon Private Network setup[30]	8
5	Resource and vCenter Management, allocating CPU resources. Source [37]	9
6	Architecture of the deploy module	15
7	Deploy build flow	18
8	Web Interface form for deployment of an instance	20
9	Status page of current build	21
10	Monitorix user interface	22
11	Deeploy command-line back-end interface for help	27
13	Deploying Ubuntu through back-end interface, time taken before and after	27
12	Deeploy command-line back-end interface for new machine - help	28
14	Deploying Centos through back-end interface, time taken before and after	29
15	Deploying Debian through back-end interface, time taken before and after.	30
17	Home screen and creation form	32
18	Build Stages Overview	33
19	Dashboard sample screen	34
16	Monitorix view from the web interface	47

4 Introduction

As of the year 2016, there are currently three billion people that have access to the internet [15]. Google handles between two and three billion search queries per day [31]. Dealing with so many requests on daily basis requires full utilisation of the available hardware resources. Part of Google’s ability to scale and be efficient is due to the emergence of cloud infrastructure. The topic of this paper deals with one of the building blocks of cloud computing - virtualisation.

”The quickest and cheapest method to providing the necessary level of abstraction in terms of server resource is currently virtualisation...”

— Paul Robinson, Google Cloud Computing [39]

The term virtualisation is defined as the ability of one piece of hardware to run multiple operating systems [17]. In this paper, a virtual machine, or an instance, is an operating system that runs on top of a ”physical” operating system. The ”physical” [operating system](#) is referred to as the ”host” system. Creating a platform that uses such technology enables an organisation to quickly set up any environment (operating system) that can be used in a variety of cases. It is important to note that there are two forms of virtualisation, bare-metal and hosted [10]. This work uses the latter approach. The difference is illustrated in figure 1.

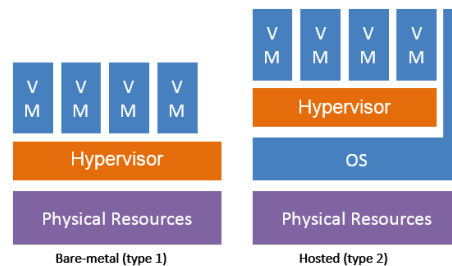


Figure 1: The two types of virtualisation [10]

When a business wants to buy a high performing computer for each employee, that would requires physical access to perform repairs, maintenance and

management. Physical systems are more difficult to manage because they are distributed across a large area, making quick access slow or impossible. Another downside of not using virtualisation is non-scalable hardware utilisation. This is the case where one machine uses maximum resources but another one is idle. A computer that has predefined hardware specification cannot use more processing power than initially installed. Automating physical servers is also nearly impossible due to them lacking the necessary abstraction control layer.

These are some problems that virtualisation technology can solve. It is now a core part of most new desktop Intel chips and is integral part of all server-grade processors as listed on Intel's manufacturing page [32].

This helps with performance, as a virtual machine can be configured on the fly to use flexible amount of resources or a shared pool of computational units. This technology also allows for easy server migration. A physical machine cannot be moved in a couple of minutes to a different continent (physical location) but a virtualised operating system can be "copy-pasted" at will to any location for very little cost. Physical infrastructure is also prone to hardware-related bugs. This is a case when a software solution causes issues, because it is not tested on all the machines that run it. These physical machines might have different network cards, processors, lack certain hardware components or have older driver versions. The virtual platform has the benefit of abstracting away these components, ensuring that once tested, the software running on top of it would exhibit predictable behaviour [24][p.59].

How does virtualisation allow for easy [migrations](#)? It achieves the task by abstracting away the available physical storage medium. On the virtualised host, they would appear as a large pool of [partitions](#) with the data ready for management. Figure 2 shows a real world application, where more storage device can be added and the current one can be assigned/managed.

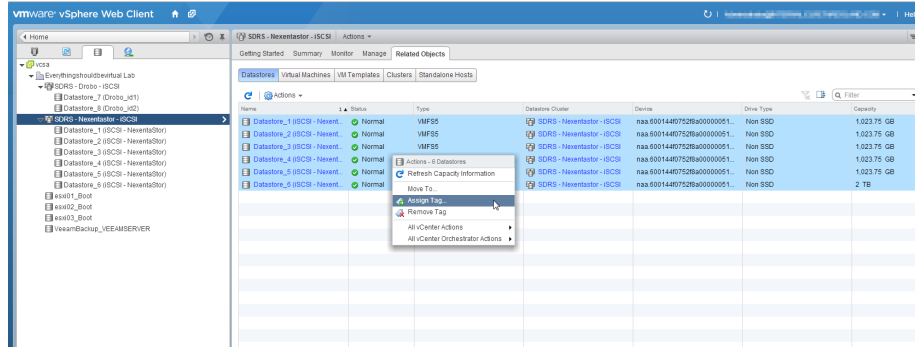


Figure 2: VSphere data-store manager screen-shot. Source [18]

Having access to the virtualisation host allows the data of such system to be copied all at once to a different provider with the only requirement - have enough capacity to store the information. The alternative without using a virtualisation platform would be to gain physical access to each storage medium individually and copying the data incrementally.

4.1 Purpose

The project aims to utilise open-source virtualisation technology and make the process of managing and creating virtual machines automated through a web interface. The solution should allow a system manager to open a website, fill in a web form with enough information about the desired [operating system](#), click a button and create it. After that, the person should be able to securely download identity file and connect to the machine.

The management website should also allow for the selection of predefined software packages. The solution should also show if the virtual machine is alive and allow the user to shut it down and bring it up. Port management should also be possible when building the machine. These features, alongside the benefits of virtualisation should create a secure infrastructure for many applications, from virtual office workstation to server testing and deployment.

4.2 Demographic

This solution is aimed at small companies who want to get the benefits of virtualisation, as well as easier machine management. It does so by giving the user common Linux versions, simple credential management and instructions during operations. The target demographic is a business who has low budget for automation and system management and cannot afford to have a dedicated team member for that task. Another use case is individuals who want to perform calculations on a machine and then power it off, who do not have resources or the time to set up their own environment.

4.3 Aim and Objectives

4.3.1 Aim

Give developers a platform for easy deployment, management and monitoring of virtual servers.

4.3.2 Objectives

Deploy Deploy a virtual machine of the user's choice through a custom framework

1. The main feature of the solution is the deployment of the virtual machine instances. The following will be achieved by using Oracle's virtualisation documentation for Virtualbox and the shell scripting language, automation tool [puppet](#).
2. The above will be accomplished through the custom ruby module written for this dissertation - **deeploy**

Firewall Configure firewall settings

- Will be achieved through the virtualisation technology's [API](#). The purpose is to add extra security layer to the guest [operating system](#). Firewall settings are managed by a software called ufw - Uncomplicated Fire Wall

Access Allow console access and set up authentication credentials (SSH keys) for the instances

- The solution will generate unique shell credentials for each machine individually. Upon creation, the user should be able to download the credentials and use them to connect to the instance.

Health Monitoring Monitor 'alive' status of the virtual instances and also show machine resource usage

Package install Allow the user to install software from a predefined list

Web Interface Create a website that will manage and instantiate the machines on the behalf of the user

5 Background

5.1 Virtualisation and "The Cloud"

"The most important feature of virtualisation that enables its cloud capabilities is the level of abstraction it provides. The key feature is the ability to hide the underlying hardware architecture." [24][p.59]. The article Heterogeneous Cloud Computing: The way forward considers an architecture that interleaves different hardware components together, enabling their utilisation on-demand. For example, a cluster of graphic cards or processors that support different [operating system](#) architectures (x86, ARM, ATOM) can be used in domain-specific tasks. Such set-up would allow the platform to be more flexible and less congested because these dedicated components would do the "heavy lifting". It also has the benefit of doing special computations - like rendering video much quicker and efficient by delegating the work to those special components.

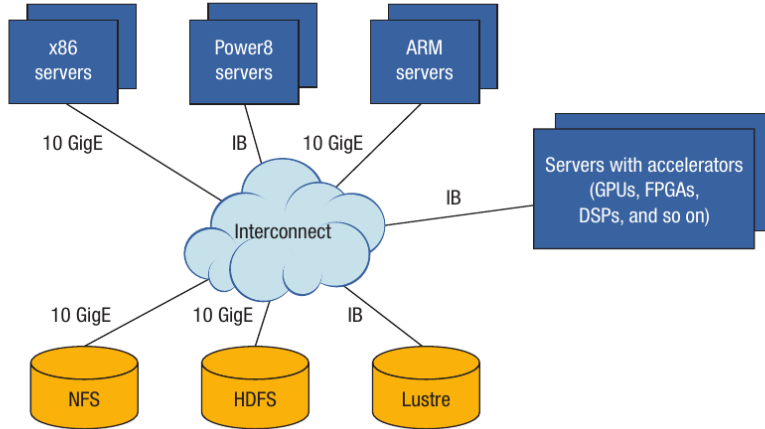


Figure 3: Heterogeneous server configuration. Source [24][p.59]

"Virtualization abstracts compute resources typically as virtual machines (VMs) with associated storage and networking connectivity. The cloud determines how those virtualized resources are allocated, delivered, and presented. Virtualization is not necessary to create a cloud environment, but it enables rapid scaling of resources in a way that nonvirtualized environments find hard to achieve." [4][p10]

It provides the abstraction that makes it much easier to get data to and from the cloud. This technology is the basis for creating high availability services and applications when implemented properly. [4][p10]. Intel's virtualisation

and cloud computing guide talks about it as a way to provide corporate platform services in a more organised matter. The guide also covers making the infrastructure easier to manage from the standpoint of I.T.. This is the case because these virtual servers are easier to diagnose and monitor as well as organise and automate [4][p8]. Virtual machine diagnostics and monitoring is a practice where the behaviour of the machine is observed over a period. This includes network traffic, memory usage, CPU performance and utilisation. It also contains information about how much disk storage the machine is taking and information about the machine's running status. Additionally, x86 computer architecture support hardware statistics counters. Performance counters are part of the kernel's sub-system and are the basis for providing reliable hardware information for monitoring and reporting [26]. This data can be used to gain insights into why a machine is malfunctioning. Counters can be vital in determining the type of issue with the machine (network related issue, hard-drive read-write failure, defecting processor, etc.). This is possible because a frozen counter indicates a failure or malfunctioning. Network outages can be detected and accommodated for by analysing aggregated network statistics. This is useful in an event of denial of service attacks (DoS). High networking usage might indicate malicious behaviour such as artificial congestion - an attacker making numerous requests to slow down the service. On the other hand, this type of monitoring can ensure on-demand scaling in critical moments. Legitimate customer demands must be met by allocating more hardware and network bandwidth as a result.

Virtual machine organisation is the practice of managing a variety of components associated with the platform and its resources. The I.T. department of an organisation is in charge of managing the physical storage on which the "virtual" platform is running. In the infrastructure disks are partitioned and organised in pools - then mounted over the network. This configuration uses S.A.N, or storage area network. This is a network entirely dedicated to transferring data between computer systems and the storage drives [33][p.11]. Mapping physical drives to virtual ones over the network is a stepping stone in enabling any type of server infrastructure, including virtualised one, to have enough capacity to grow and meet the customer's demands. It also ensures the right level of redundancy, as a large portion of the hard drives can be put in place in case of a failure. Storing virtual machine on network drives makes it possible to copy machine instance from one cluster to another, or change data centres.

Any server infrastructure requires high bandwidth network connection, virtual networks are no exception. The network throughput of the platform depends on the internet service provider (ISP) physical connection speed. That network is used by the virtualisation platform and is converted to numerous private and public networks internally. The benefits of such abstraction is the ability to create flexible and disposable internal/external networks per machine or cluster of machines. The ability to share a network, or be in a private network is a security consideration. This feature is important from the standpoint of security. For example, for a business logic application, one can set up a client-facing web server that is accessible from the global internet. The machine on which this application is running will have a database connection

to another machine running a dedicated database software (mysql, postgresql, mongodb, etc.). The connection between the two machines would be configured as a private network. It ensures that nobody from the outside world has access to that database machine. This configuration is explored in Amazon's accessing database instance in a virtual cloud [30]. The setup is shown in figure 4.

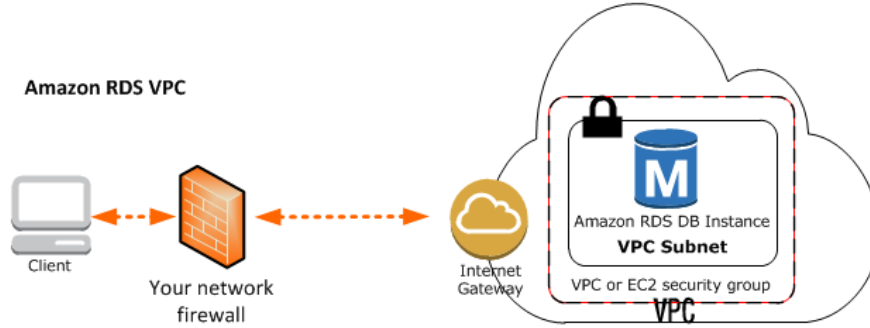


Figure 4: Amazon Private Network setup [30]

It is impossible to talk about such infrastructure without considering the most important component of operations, the **CPU**. When providing the hardware resources for computing, having enough calculation power is essential, otherwise each virtual machine would be slow and unresponsive. A server rack with many IntelTM Xeon or other server-grade chip are set to run on top of the virtualisation technology (virtualbox, VmWare 1. All of the computing resources are then also abstracted away. For example, when using VmWare vSphere, instead of seeing every single processor, you are given the option of allocating certain amount of megahertz per instance. A guide on configuring CPU usage is covered in VMware Cookbook [37]. CPU allocation screen is shown in figure 5

Considering the fact that all of these hardware components have to be managed by a human to ensure smooth operation during heavy and light loads, automation plays very big role in the process. Automation is the process of automatically creating virtual machine and dynamically allocating computing resources associated with it. Big cloud platforms such as Amazon Web Services and Azure have almost absolute automation. Azure even provide their tool as a service - Azure Automation [9], Amazon use tools like chef and in-house tools [23]. Computing resources are allocated based on usage, where a paying customer can use a lot of resources for a certain price (scaling on demand).

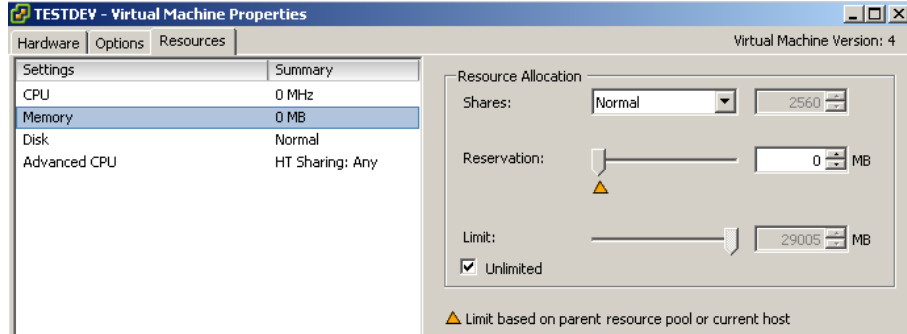


Figure 5: Resource and vCenter Management, allocating CPU resources. Source [37]

5.2 Similar in the field

For the duration of this work, a variety of technologies, frameworks and utilities have been evaluated, that will achieve the goal of automated system deployment. Part of the work was experimenting with different "cloud" virtual machine deployment services. This has given insight into the key requirements that ought to be provided by the platform.

Something that all major platform providers have in common is the ability to scale and their price model. The reason is that the platform allow for the hardware configuration parameters to be tuned at will per user within minutes. This means that when the [virtual machine](#) is created, later on the "hardware" resources can be modified and changed easily.

A commonality is the price model. All of the below mentioned platform providers use some form of pay-as-you-go. This means that companies requiring extra computation at time critical periods can take advantage of it. An example would be personal project of Harrison Kinsley for gene sequencing in his online learning series[19]. He used 5 [GPUs](#) for an hour to perform the necessary computation, then retrieved the results and then destroyed the machines. As a result, one can use sophisticated hardware for a fee with very little set-up overhead.

5.2.1 Amazon Web ServicesTMEC2

Amazon are the leaders in cloud deployment with a 30% share of the world market share[11]. Their services range from providing [DNS](#) to deploying their custom virtual machines. Because of the widespread usage of their service (popularity), this is the main platform that inspired the work.

The main focus of the dissertation is not about the AWS platform, but about their service EC2 - virtual machine deployment.

EC2 allows a user to do the following:

- Create scalable [virtual machine](#). This includes selecting [operating system](#) type with a variety of options (Windows Server Ubuntu Linux, Red Hat Linux, Amazon Custom Images). They also offer machines for database only purposes (Amazon RDS).
- Select from a family of options that include general purpose machines, memory optimised, storage optimised, machines with dedicate [GPUs](#) and optimised storage.
- The amount of [CPUs](#) to be used for the machine
- Amount of [RAM](#) in gigabytes to be used
- Storage capacity and storage type, e.g. how much and how fast it can be.
- Because they have their network load-balancers, it is also possible to configure the network performance of the machine(throughput).
- Configure IPV6 support. IPV6 is the updated internet protocol that allows for 3.4×10^{38} network addresses to be allocated.

Some other benefits of EC2 is the ability to integrate with every other Amazon Web Service product. The service is also running on top of Amazon's network which ensures reliability and availability across many regions. EC2's pricing model has application for hobbyists and corporations alike due to the provided hardware options and the flexible pricing. Prices range from \$0.044 to \$14.400 per hour [13]. The service also receives publicity from having Netflix and Airbnb hosted on their platform. A downside to AWS is that they do not allow for custom [kernels](#) to be used, leaving users to rely on the predefined images provided by the platform.

5.2.2 Microsoft Azure

Microsoft Azure is also among the biggest virtual machine and cloud computing providers in the world and has 10% of the global market[11]. As this is a Microsoft product, it is fully integrated with their text editor - Visual Studio. This makes it easier and more convenient to develop, deliver and integrate with Azure through the tool. Azure also has the benefit of being a very large provider - allowing them to use 34 data centres distributed across the globe [5]. This ensures high availability and fast response times. The service is also compliant with regional regulations and requirements, making it suitable for fintech and health service deployment [6].

As the platform is provided by Microsoft, the creators of Windows, it can provide [operating system](#) specific features to its customers, such as integration with their products and services. This makes it ideal for Windows compliant companies that have bought licensed corporate software. Similar support would be difficult from a different provider, as they are not the maintainers of the

operating system Windows. Users of [Azure](#) also have the benefits of Microsoft Windows features coming first to their platform.

6 Work

6.1 Overview

This chapter aims to give a more in-depth view into the process of scoping, developing, testing, creating the architecture and building the application. This ranges from picking the tools to limiting limiting or expanding requirements, the assumptions made, as well as a break down of each task.

Creating such a platform from scratch is a daunting task even for an experienced software engineer, that is why the work contains various third party modules and software.

For the above reason, the initial research into building the right tool-chain was essential to the success of the project.

The development methodology can be broken into the following steps:

1. Research into virtualisation technology to understand the underlying components.
2. Select free and open-source software components that can be coupled together to orchestrate and utilise above mentioned technology. The public libraries used also allowed to extend, modify and build on top of it during the development process.
3. Build a set of core and advanced features. Core features are essential for a good operational and usable solution. Advanced features are the things that make this solution go above and beyond the scope of the base features.
4. Integrate components into one final solution
5. Do sufficient testing to ensure that the platform behaves as expected.
6. Build a web interface that ties the creation process into a web form
7. Allow for health-monitoring per machine instance

A hypervisor is a computer software or hardware that creates the platform layer to a virtual machine and makes it possible to host multiple instances with different configurations [22].

Initially, the plan was to use a hypervisor virtualisation solution such as [virtualbox](#) and [VmWare](#) just by themselves, and use the [bash](#) shell scripting language to interface directly with them. A decision was made to use Virtualbox, as it is an [open-source](#) software solution, has strong community, allows third party plug-ins and is a solution build by Oracle - a large software corporation. There are other alternatives (gnome boxes, CHROOT), but what influenced the choice is that virtualbox is configurable and can be controlled from the command-line straight out of the box. Another deciding factor was also the extensibility that it offers with third party modules developed by the community and the core team (guest additions, virtualbox standard development kit).

A reason for not choosing a more unpopular technology is due to the willingness of an adopter to switch their entire virtual infrastructure to it. The Virtualbox software is well supported by Oracle, and as such, it would make the transition to the platform easier.

During the period when researching Virtualbox, the command-line [API](#) documentation was used extensively. In the process, a variety of customisable parameters when building a virtual machine were discovered, like the different possibilities in setting up networks (private, network shared, host only, etc.), exposing ports, as well as different hardware configurations (RAM, hard drive, CPU count). After using the command line tools and writing a simple shell scripts for managing creation of different resources, a decision was made to use a tool that manages the different technical details of each virtual machine. Instead of building the manager from scratch, an open-source hypervisor provisioner software was used. The tool is called [Vagrant](#).

6.2 Vagrant

Vagrant by Hashicorp, is a tool built by a company specialising in easy virtual machine configuration management and set-up. Their [API](#) provides the following features.

- Allow the administrator to create a [Vagrant](#) configuration file (called Vagrantfile), which describes end-to-end each operation that will be performed to the virtual machine instance (including the Linux distribution that will get installed on it).
- Create a system compatible with their platform starting from a bare configuration and incrementally building a bespoke instance. These instances can be managed and shared on the internet.
- Managing network interfaces
- Configure ports
- Allow port access to adapter network (in private networks)
- Create a custom machine compatible with the platform
- Kick-start automation by running one of the supported automation frameworks (chef, puppet)

6.3 Workflow

The framework that was created uses a [virtual machine](#) instance object and authenticated user that owns it. When the user fills the web form with valid information, it will pass the configuration values directly to a virtual machine class. This class contains a dynamic configuration, which ensures that the solution is distribution agnostic. Here, the term distribution references to the

different versions of Linux (Ubuntu, fedora, Centos, arch Linux). Separating dependency specific configuration by sub-classing a configuration object allows for a quick way for the project to be extended and accommodate for new distributions.

To do so, folder with the name of the new distribution must be placed in `lib/configurations` that describes the configuration specific settings - including updating and installing common software packages.

The framework also uses UNIX environmental variable to manage infrastructure state. For the purpose of continuous integration, two main isolated environments are used - testing and development/production. The most notable usage of such environment is the virtual host-only network that ensures that network interfaces do not clash during testing and development. The two interfaces are `vboxnet0` and `vboxnet1`. One is for general purpose development, the another is for testing. A special host-only adapter is used to have an isolated ip range that is network agnostic, for example, on a [NAT network](#), some IP addresses might be reserved or used, this case will cause a clash and the virtual machine will not be reachable.

Continuous integration is a new software development practice where versions of the product are merged and available after testing/building phase. It also enables features to be developed separately from the production environment. This has the benefit of ensuring production code performs as expected without interruptions.

6.4 Languages and frameworks

When considering the architecture of the system, a list of factors were evaluated when building the solution. One is having few well consolidated moving parts, ensuring flexibility and extensibility. For that purpose the programming language [ruby](#) was chosen. It is a suitable for system automation as it is very expressive and is for the most part platform independent (interpreted). Its suitability comes also from the available tools [Vagrant](#) and [puppet](#) which allow end-to end integration when developing and testing. The [ruby](#) programming language also draws inspiration from the Perl programming language. One of the Perl's strengths is its ability to be a more powerful shell scripting language as well as a general purpose language. This is the case as it has quick syntax for executing shell scripts directly and get STDOUT, STDERR and exit-codes quickly. It does also has native implementation of common UNIX tools, like `grep`, `wget` and `awk` (core GNU utilities).^[2]

Because the Ruby language draws inspiration and shares similar features with Perl, it makes it a great modern and powerful language to use in this domain-specific problem. The language benefits from it being written in 1995, because of that it is very mature. For that reason, there is a great amount of online help and third party stable libraries.

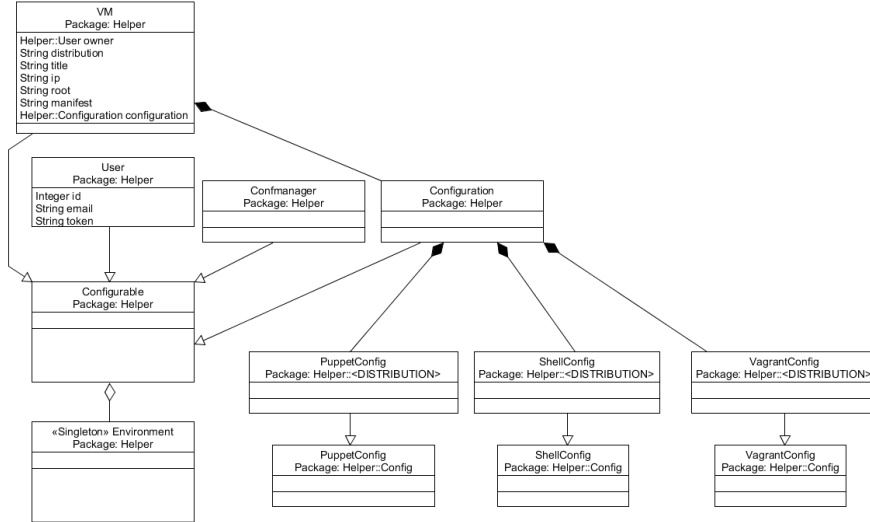


Figure 6: Architecture of the deploy module

6.5 The deploy module - architecture

The first component of the work is a ruby module designed specially to act as a back-end for the entire solution. A module is a collection of programming functions and libraries that are bundled together to achieve a particular task. In [ruby](#), modules are called gems. For writing my first [gem](#), the official ruby guide was used as a starting point [36].

The file `lib/deploy.rb` is what sets up the database connection and bootstraps the whole module by auto-loading every component (class). To ease integration, the file loads configuration properties from `lib/config/application.yml`. `Application.yml` contains settings for production, development and testing environments, this is to allow separation and configuration of different parameters. The file specifies location of the database file when in development, or it has the database drivers in production with information about connection parameters. The file also describes the network interfaces to be used in each environment.

`lib/deploy.rb` has a helper method that returns the current network interface and `network mask`, this function is responsible for generating [IP addresses](#) and also for verifying that the virtual network is up and running. It does so by issuing commands for bringing up the virtual networks `vboxnet` up, then using network sockets to extract relevant information. This function is the programmatic equivalent to the command `ifconfig` or `ipconfig` on Windows and Linux.

The file also contains a static function that returns a map of all supported

distributions and the associated [Vagrant](#) machine. The function is helpful when validating creation, and will error out if the [distribution](#) name is not in the list during build. When extending the supported distribution list, this function needs updating.

Similarly to the supported [distributions](#) function, there is a static function to return the supported packages list, used for displaying the options and validating unsupported packages.

The module provides a function `slugify`, its purpose is to convert string with special characters and spaces to a set of words and dashes, non-ASCII characters are stripped.

[IP address](#) helper function checks if the supplied is part of the current network by using ruby's `ipaddr` build in module.

The file `lib/vm.rb` contains the code for managing the virtual machine instances, the class is called `Deeploy::Configurable::VM`. The class inherits from `Configurable`. Calling `VM.new` contains validators that raise errors upon initialising the class inappropriately. The method is responsible for specifying the distribution, available resources, configuration destination, name of the instance, packages, open ports and prepares it for creation. A design decision was made to allow a machine to be configured with the `new` operator and then created by issuing `machine_instance.build()`. The reasoning is that between the two stages, validation and injection of dependencies can be done.

The method `VM.alive()` verifies if the machine is alive by trying to listen on [SSH](#) port, if it fails within a time-out, the machine is updated to state of not alive.

`VM.get()` is used when performing power up, power down and destroy virtual machines. It queries the database and discovers everything about the instance, from packages, to [RAM](#) and location of configuration files.

The virtual machine is built only when an instance is initialised properly by calling `VM.new` with the correct arguments and then invoke `VM.build()`. The design decision to have a separate functionality of bootstrapping instances was mainly for cleaner testing by compartmentalising the components into smaller elements. The `build` method also accepts a boolean argument. This argument tells the current build if it is in "dry-run" mode. In this mode, the machine will create directories and configuration without actually bringing the instance up and running. This is again done for helping during tests are running - they verify that different [distributions](#) have appropriate configurations.

During the invocation of the `build` method, a [bash](#) shell is forked and it executes the [Vagrant](#) command that brings the instance on-line and installs all dependencies. The shell command also tells the script to write all output to a text file in the machine directory, the file is called `vagrant.log`. Because the process takes a long time to finish, an axillary function `_wait_on.build` is used. This function constantly goes through the contents of the log file and checks what is the current building state of the instance. Typical states are building the base, installing puppet, setting up dependencies, installing packages and the state everything has finished. Upon a successful build, the sub-shell executable will create a file called `build.log`. The file will contain a build

status. Reason for doing so is that otherwise successful run is determined by exit-code of vagrant, which is not a reliable method of determining success.

The module also partitions the file structure in two parts - **userspace** and **test.userspace**. The concept of userspace is used to contain virtual machine configurations in a hierarchical structure. Inside, there is a folder representing each user registered through the web interface. Inside the user folder, virtual machine folders with their respective names can be found. Each virtual machine folder contains **Vagrantfile** - file that describes network configuration, machine type, available hardware resources, etc. The folder also contains **vagrant.log**. This contains detailed information of what happened during the machine build process, and is the file that a user can view through the web interface.

In **manifests**, two very important files can be found. One is in charge of automating firewall rules, packages, users and permissions. This automation is done once the machine has been brought to live. The file is created by the configuration manager depending on which Linux distribution has been used. This folder also contain a shell script that can be used when automation script is lacking. This file has been left there for integrity. As mentioned, there is also a top-level folder called **test.userspace**, it is used for separation of concerns and is where testing machines are deployed to keep production machines intact.

The process of building a virtual machine by the **deploy** module is also illustrated in figure 7.

6.6 Virtual networks

Because each machine has to be connected to the outside world, network **IP address** has to be allocated to each instance. This task is achieved using **virtualbox**'s host-only adapter. The gateway address of the adapter and its **network mask** are obtained and a list of possible **IP addresses** are generated. Out of that list, all reserved ones are blacklisted. Additionally, the database gets queried for a list of already allocated IPs. After the process, the machine picks the first address from a stack and the value gets used. The reason for allocating them sequentially starting from the top is to avoid collisions. Other methods caused issues where the algorithm had to be run multiple times to obtain reliable results.

After generating the address, it gets blacklisted for further use, ensuring uniqueness on the network. When a machine is destroyed, its address becomes available again.

6.7 Extending virtual machines

The process of creating a virtual instance is using a custom image that has a variety of pre-installed software and firewall rules specially created for the dissertation. The purpose is to increase deployment time and allow for a more controlled package list.

Then a friendly name to virtual machine instance map entry has to be added to the **deploy.rb**. After that, a folder with the name of the new distribution

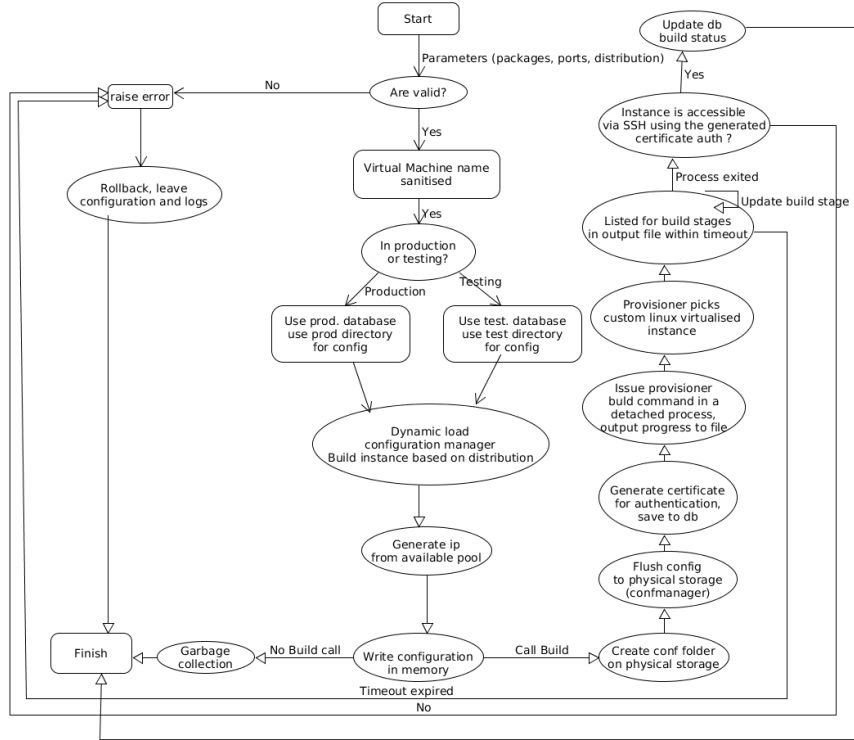


Figure 7: Deeploy build flow

must be placed in configurations containing files who subclass `firewall_config.rb` `packages_config.rb` `puppet_config.rb` `shell_config.rb` `vagrant_config.rb`. These files might be left blank if the distribution uses the default commands. Any deviations should be extended via these child modules.

6.8 Building The Web Server

The core module `deeploy` is natively integrated into a [Ruby On Rails](#) web-server. Ruby on Rails has been chosen as it is one of the most popular web frameworks and is used by big online companies like Twitter [14]. Other reasons are that during the evaluation process, the documentation and support was extensive and provided enough information to quickly start building application with little overhead. Being written in [ruby](#) made it an easy choice when integrating.

6.8.1 Database and authentication

First part of building the web server was creating relational database model for different machine states and building a credential system for user authentication and authorisation. Password concerns are described in section Passwords and authentication.

6.8.2 Web Form

The web interface has a web form where machines can be build. Every filed in this form is validated on the back-end to ensure proper passing of user parameters. The validation happens transparently through the `deploy`'s helper functions. Upon detected issues, such as open ports outside of available ranges, conflicting instance name, invalid packages, logged user and supported distribution, the web form gives hints of what went wrong. The form also validates compatible packages, for example package `nginx` and `apache2` cannot be installed simultaneously as they work on the same port. The process is illustrated in figure 8.

6.8.3 Asynchronous deployment

Initially, the interface had an issue with instance build responsiveness. It took the command to build a machine and it waited until the whole process was completed (1 to 2 minutes) before the success screen was shown. This is the case, as the web request will not complete until the `deploy` module returned an exit code to the build process.

To avoid long wait times and unresponsive interface, redis cache queue is used. Redis is a durable in-memory key store. Upon build request, it returns a success code and a loading screen is shown (figure 9). Then the build request is placed in a queue in `redis` memory. A build worker is implemented to monitor for any pending redis jobs and picks them. Once picked, the build process begins. During that stage, the `deploy` module updates the database machine object's build progress. The website's front end constantly pings the [API](#) to obtain the current build stage, if the stage is success or failure, the continue button appears. The same pipeline is used for stopping, starting and terminating a machine.

Upon unsuccessful build, the web server pulls the build logs and creates a downloadable link to the resource. This is done for debugging purposes. The middleware which ensures that `redis` jobs are completed is called `sidekiq`, the build worker discussed above is written with the support of `sidekiq`'s documentation [25]. It is important to note that if `redis` is not working, the machine build functionality will not work. For this reason, in order to run the web server, `redis` and `sidekiq` must be up and running. Code has been added to raise exceptions when these tools are not running.

Deeploy

* Title

concreting-twelve-concreting-crosswind

* Vm user

user

Ports, comma separated

Distribution

ubuntu

RAM in Megabytes (Random Access Memory)

500

Packages

☐ vim
☐ nginx
☐ apache2
☐ mysql
☐ memcached

Create Machine

Deeploy

* Title

crosswind-concreting-rootage-unmounted

* Vm user

can't be blank

Ports, comma separated

asd

Ports must be numbers

Distribution

ubuntu

RAM in Megabytes (Random Access Memory)

1111111

Random Access Memory must be between 256 and 1024 MB

Packages

☐ vim
☐ nginx
☐ apache2
☐ mysql
☐ memcached

Cannot install nginx and apache2, conflicting packages !

Create Machine

Figure 8: Web Interface form for deployment of an instance

6.8.4 Implementing the Health monitor

Two strategies were used when checking machine status. The first one is using a health service 'worker' process. This is again written in with the help of `sidekiq` and has hooks with `deeploy` module's machine state. A `CRON` task runs every minute and checks the state of each machine via the worker. A machine is assumed to be alive if the health worker gets an acknowledgement after connection attempt on `SSH` port 22. If this connection cannot be established, the machine is assumed to be inaccessible.

The second strategy is pre-installing a free and open-source software `monitorix`, a tool for monitoring a variety of hardware parameters over a large period. The setup is configured in such a way that a firewall is blocking external connec-

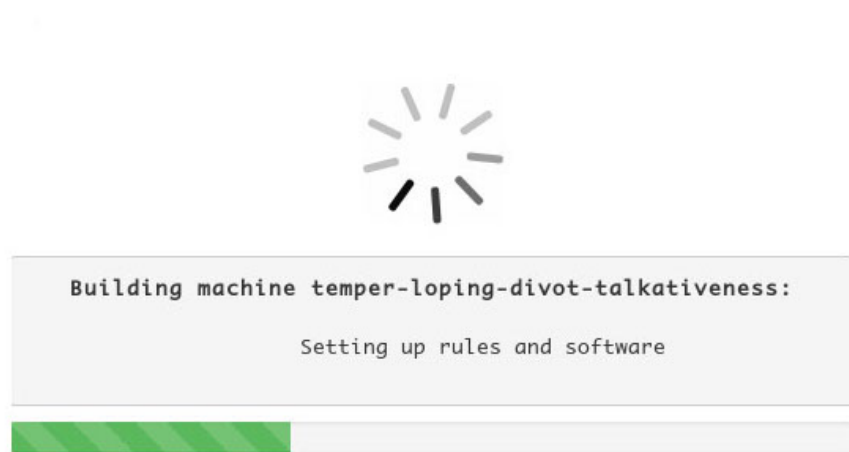


Figure 9: Status page of current build

tion to the monitorix's ports and only the web platform's [IP address](#) is allowed. This ensures that only a logged-in user can view the performance statistics of the machines. A running interface of monitorix is shown in figure 10

6.8.5 Obtaining Certificates

The web server has the necessary code to allow an authenticated user to download their authentication certificate files for gaining access to their respective machine. Work has been done to ensure the user understands how to use the certificate - a short code snippet tailored for the machine is provided with each instance. Upon certificate download, the authentication files are deleted from the server for security reasons.

6.8.6 Management Functionality

The interface is also a gateway to managing the state of the machines. The web dashboard of a logged in user, as shown in figure 19, gives information about many aspects of the created instances. This is where the management panel for powering on, off, destroying and restarting is exposed. All of these features use the core `deeploy` module for this functionality.

6.9 Security

Virtual machine instance isolation, privacy and authorisation is one of the most important security concerns. This section describes the decisions and considerations made during the development process. A variety of architectural and



Figure 10: Monitorix user interface

operational choices were also made to ensure that upon security breach, little or no data gets leaked.

6.9.1 Passwords and authentication

The system makes the assumption that the management software is not compromised, as this is where all user credentials and data is stored. Access control is done via relational database and every machine command is authenticated against it.

For password disclosure mitigation, the passwords in the database are not stored as plain-text. Instead, the `bcrypt` hashing algorithm is used with salt and pepper. In production, the operation is done for 11 rounds. The number of round is a configurable parameter.

6.9.2 SSH Credentials without password

A decision about access to the machine is made in favour of public-private key cryptography instead of using password. This improves security, as it generates unique pair per instance, and eliminates the risk of forgetting the password or

having someone steal or guess it. Two strategies were considered - allowing users to import mandatory public key they initially generated, or generate key that the user can later download. The second approach was selected, as it would make it much more convenient and intuitive for the demographic of the platform. Future work should allow privacy concerned system administrators to generate their own keys and disable any keys generated by the system.

6.9.3 Guest Isolation concerns

This is one of the greatest benefits of running in a virtual environment. All the machine operating on top of the platform are isolated from one another and one [operating system](#)'s calls cannot interfere with another. In terms of architecture, this is a good design, but there are some drawbacks.

One of the most severe issues with guest isolation is exploiting a vulnerability in virtualisation memory and reading another machine's memory. This attack is known as "Virtual PC Hypervisor - Memory Protection violation" [29]. An exploit can cause passwords leak and disclosure of sensitive information.

6.9.4 SSH Keys

The tool [Vagrant](#) has two modes when generating access keys - insecure and secure. When insecure keys is used, every machine has the same credentials. The second configuration is used, as exploiting one instance leaves the whole infrastructure breached.

Furthermore, once the [SSH](#) keys are downloaded by the client, they are cleared from the host system.

6.9.5 Source of packages

The solution does not provide modified version of any software - the reason is to reduce overhead of maintenance cost. Even though the machines have pre-installed applications, they have not been modified. This approach makes it possible to easily update any package from its official maintainer instead of by the platform vendor who would not necessary be expert in package management.

6.9.6 Development practices

This work has been developed using the official documentation for every framework or language. The goal is to minimise exploitation vector by following the instruction of organisations, as they have greater insight into best practices. This makes the solution easier to maintain.

The work-flow is supported with the version control tool Git, which is an industry standard in managing software. Using the tool allowed for a main branch where all stable features were merged. Each new feature was developed in a branch, a term that references to having an isolated copy of the source code containing the new bits of functionality.

Continuous integration methodologies were adopted, and the solution benefits from having two separate environments for testing and deployment, ensuring normal operation during new features (lack environment conflicts due to isolation).

The object-oriented pattern for building modular software was used. This approach ensured separation of concerns - each file does one function and it does it well. This approach makes it possible to extend the functionality of the platform easily.

7 Testing

One of the concerns when developing the platform is testing interoperability. Due to the many moving parts, a subtle change might break the whole system. For that reason, a variety of automated, manual and built-in tests are ran periodically.

Most notable, the core `gem deploy` contains unit tests covering every element of the machine process. The official best practices guide was used to bootstrap the process [28]. The `ruby` unit test framework is called `minitest` and is the recommended solution from the framework provider. The core tests verify that an address can be allocated, machine name can be generated and try to perform basic operations, such as building every possible machine. The tests also try to install every package, open a variety of ports, bring machines up and down and destroy them.

As a core principle, the testing is implemented in a such way that is self contained and clean-up is done after each test. This ensures that tests do not conflict with each-other and testing is compartmentalised. During a running of a test, before and after actions are always triggered to bring the environment to a known state.

The tests also set a testing environmental variable, which forces them to be on a testing private network with their own name-space. For example, in production, the configuration of users is stored in a folder `users`, but in testing, a different folder called `test.users` is used. This separation ensures both environments can coexist without overwriting each-other or conflict on the same network. The web server also uses a separate testing database during testing to ensure no users become affected. This methodology is achieved thanks to the initial architecture of the solution - as it is always aware of which context it is running in.

The unit tests had a very positive effect on the quality of the development process. When a new feature is implemented, it is pushed upstream only after the tests are passing. Another benefit of the written tests is their integration into the core of the product. For example, the testing if a machine is running after deploy is performed every time upon build, if the test fails, the actual deployment fails. Having automated tests also reduces drastically manual testing, giving more time to do actual development and implement features.

Even though the core component `deploy` has automated tests, the web side relies heavily on manual testing. When any new changes are made, the whole interfaced is checked for errors by hand to ensure proper integration. Generally, this is not a good practice, but it was a compromise due to little development time. If more time was available, selenium web driver would be used to do full interface testing. Selenium allows for Javascript - client side code to be executed on a web page, making it possible to automate user clicks and actions. The issue is that integrating this testing framework with the solution would have taken significant amount of effort and time.

8 Evaluation

8.1 Introduction

The evaluation of the work has been done with the help of Newcastle University [I.T.](#) department. They helped with identifying potential security and usability issues as well as give general feedback. This chapter contains a detailed overview of their evaluation and also personal reflection about the achieved tasks. Overall the solution meets the specification requirements. Improvements can be made, as this is the first version of the platform and a perfect solution cannot be achieved from the first iteration.

8.2 Deployment Overview

The requirement Deploy is listed on page 4 in section ‘4.3.2’. The objective of deploying a virtual machine has been achieved by giving the user the opportunity to deploy 3 distinctive distributions. The process takes between 1-2 minutes and requires an authenticated user.

When using the back-end tool, it is possible to do everything that the web interface can. Issuing `deploy --help` and `deploy new --help` brings up a help menu that gives a detailed command line manual page. Figure 11 and 12 show the output.

The interface thoroughly gives information about how to achieve a task with examples. This way, manual deployment can be done with minimal knowledge and lack of documentation by someone unfamiliar with the system. Creating this feature costed large amount of time that could have been allocated to a different feature. The reason is that the tool would rarely be used manually and all the work would end up unnoticed.

8.3 Deployment Time Results

The time it takes for a machine to build is critical, the project aimed to allow the user to quickly create a machine and be notified upon completion within sensible time. The deployment time of Ubuntu and Debian takes 1 minute (figure 13), Centos takes between 1 and 2 minutes (figure 14 and 15).


```

NAME:

    deploy

DESCRIPTION:

    Create and destroy virtual machines

COMMANDS:

    destroy Removes a virtual machine
    down    Turns a virtual machine off
    help    Display global or [command] help documentation
    list    list virtual machines associated with a user
    new     Creates a new virtual machine
    up      Turns a virtual machine on

GLOBAL OPTIONS:

    --auth_user STRING
        Specifies the authenticated user

    --auth_password STRING
        Specifies the authenticated user's password

    --auth_token STRING
        Instead of using user/password, use token instead

    -h, --help
        Display help documentation

    -v, --version
        Display version information

    -t, --trace
        Display backtrace when an error occurs

(END)

```

Figure 11: Deeptool command-line back-end interface for help

```

deploy new --distribution ubuntu --vm_user=plamen
--auth_token *****

Sun 30 Apr 18:45:50 BST 2017
Building VM 'callipygian-engaging-reposition'.
Please wait
Status: Configuring network interfaces
Status: Setting up rules and software
Build successfull !

Operation successfull:

VM NAME:      | callipygian-engaging-reposition
IP:           | 88.168.56.254
User:         | plamen
Distribution: | ubuntu

Sun 30 Apr 18:46:41 BST 2017

```

```

NAME:
    new
SYNOPSIS:
    deploy new [options]
DESCRIPTION:
    Deploys a new virtual machine, sets up user and allocates public ip
EXAMPLES:
    # Example
    deploy new --auth_user usermail@website.com --auth_password password --vm_user nameofvm --distribution ubuntu
OPTIONS:
    --vm_user STRING
        Username of the machine
    --distribution STRING
        The distribution flavour (ubuntu, debian, etc.)
    --vm_name STRING
        The name of the machine, generates a random name if left blank
    --ports STRING
        Ports that need to be opened, comma seperated
    --packages STRING
        List of software to be installed on the machine, comma seperated
    --ram STRING
        Size in gigabytes for the machine

```

Figure 12: Deeploy command-line back-end interface for new machine - help

A deployment of a machine took more than 5 minutes initially, but work went into reducing that time by packaging common applications together. This approach was successful and reduced client's wait time significantly, making the tool more usable.

The benchmarking numbers are achieved on Intel(R) Core(TM) i3-4170 CPU @ 3.70GHz, ATA Samsung SSD 850 EVO drive. Machine running Fedora Workstation 25 Headless.

8.4 Deployment Progress

During build time, the deployment script outputs progress messages when key stages are detected. The system would become more user-friendly if more stages are added. Increasing response time adds to usability and perceived reliability.

8.5 Investigating Failures

Deploying a machine successfully is not always the case. Large chunk of the code ensures that errors are detected, reported and deployment is rolled back (cleanup). This is important, as the user of the system must always be aware of any progress or failure. Detecting and reporting unsuccessful builds is done via providing failure status in the web interface. The failed machine's logs are also accessible, making it possible to report issues via emailing such files to a system maintainer. The platform detects network failures, [IP address](#) allocation errors, bad arguments or build taking too much time as well as deployment script

```
deploy new --distribution centos --vm_user=plamen
--auth_token *****

Sun 30 Apr 19:46:54 BST 2017
Building VM 'thyroidal-corporate-break-of-the-day'.
Please wait
Status: Configuring network interfaces
Status: Setting up rules and software
Status: Packages Repository updated
Build successfull !
```

```
Operation successfull:
```

VM NAME:		thyroidal-corporate
IP:		88.168.56.251
User:		plamen

Distribution:		centos
---------------	--	--------

```
Sun 30 Apr 19:48:25 BST 2017
```

Figure 14: Deploying Centos through back-end interface, time taken before and after

```

deploy new --distribution debian --vm.user=plamen
--auth_token *****

Sun 30 Apr 19:49:41 BST 2017
Building VM 'unflurried-sparing-czech'.
Please wait
Status: Configuring network interfaces
Status: Setting up rules and software
Build successfull !

```

```

Operation successfull:

```

VM NAME:		unflurried-sparing-czech
IP:		88.168.56.250
User:		plamen

Distribution:		debian
---------------	--	--------

```

Sun 30 Apr 19:50:32 BST 2017

```

Figure 15: Deploying Debian through back-end interface, time taken before and after.

crashes. An issue with this process as of now is that detection happens, but detailed report of what went wrong is lacking. An extension to the system should allow the user to go to a dedicated error page. It will give a description of what exactly caused the failure and give the ability to report it.

8.6 Extensibility of solution

Part of allowing users to use any machine they want is due to the platform being extensible. As described in the work section of this paper, the solution achieves this extensibility - evident by the supported multiple machines. The issue with this feature as of now is lack of documentation. An article that describes the process of adding a new supported distribution with pictures and examples would improve adoption and ease of use for future development.

8.7 Health monitor

The requirement Health Monitoring is listed on page 4 in section '4.3.2'. Part of the specification is about monitoring virtual instance's health. This topic includes detecting machine crashes, sending log reports, restarting the [kernel](#) upon failure, monitor available resources and much more. Because creating a complete health monitoring solution would have been outside of the scope of this dissertation and would have taken large amount of time, a compromise solution

has been provided. What has been implemented is a balance between effort and usability.

The current solution uses [CRON](#) job to check the status of each virtual machine every minute. The result updates the web interface and a user can notice that a machine is running or not. The process involves checking if the [SSH](#) port is responding to packet alive messages. This port has been selected, as it is the gateway to the machine. If no response is given back, the machine is assumed to be dead or unusable.

Other than alive information, the solution provides detailed statistics about resource usage over time. It can show processor load over a large time period, disk writes, network traffic usage and many more parameters. This information is available in the web interface under each machine's own section. The implementation is shown in figure 16.

Checking the 'alive' status of a machine can be implemented more efficiently. The current implementation is prone to scalability issues, where numerous instances linearly slow down the process. The check also has a large overhead due to making requests to every possible machine (even ones that are known to be in good state). Future work can improve the process by adding a ping service on each machine, which periodically contacts the server to update 'alive' status. This approach would help with scalability and would reflect a machine's state more accurately. Furthermore, the [Nagios](#) software can be installed on each instance for even richer status information. Nagios is the industry standard in [I.T.](#) infrastructure monitoring [8]. Having this open-source professional grade software would also benefit the platform when it comes to adoption and reliability.

8.8 Web Interface

The requirement Web Interface is listed on page 5 in section '4.3.2'. Large portion of the work went into creating an intuitive web interface through which a machine is being deployed. During the demonstration, positive feedback was provided for the overall experience and feeling throughout the process. Once the user logs to the web interface, the option to create a new machine becomes available. The pages are shown in figure 17.

Deeploy

Home

Welcome, local@host.com

Sign out

Personal

API Key: securetoken

Create new virtual machine

Info No current virtual machines

Deeploy

* **Title**
fantastically-noncontending-challenged-thorstei

* **Vm user**

can't be blank

Ports, comma separated
ertftrtyffght
Ports must be numbers

Distribution
ubuntu

RAM in Megabytes (Random Access Memory)
5454545454500
Random Access Memory must be between 256 and 1024 MB

Packages
☐ vim
☐ nginx
☐ apache2
☐ mysql
☐ memcached
Cannot install nginx and apache2, conflicting packages !

Create Machine

Figure 17: Home screen and creation form

When submitting the form, few validation passes are made before the machine is build. Validating parameters is essential, or the build process will fail when receiving incorrect arguments. The form provides clues about failed validation, making it convenient to fill in the correct details. Validators check uniqueness of machine name, presence of virtual machine owner, distribution, **RAM** within range and conflicting packages.

When the correct parameters are passed and the form is successfully submitted, the user receives a progress bar indicating the build status with two possible outcomes - success or failure. The various stages are shown in figure 18.

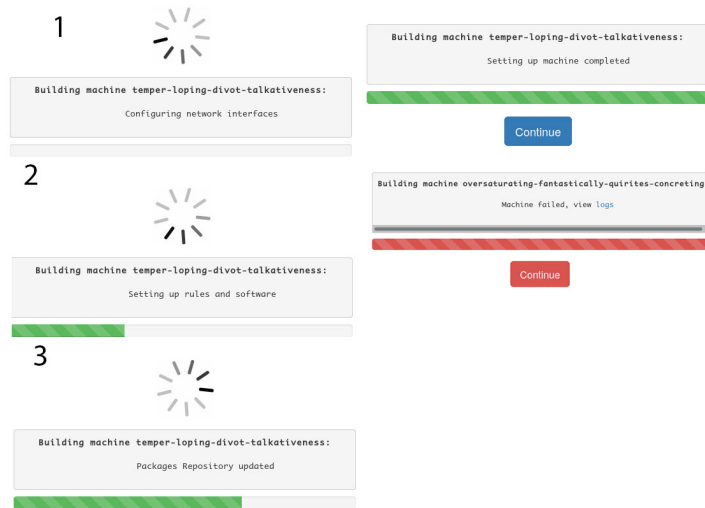


Figure 18: Build Stages Overview

Upon failed build, the logs can be checked to gain insights into what went wrong.

From the web dashboard, it is possible to view the running state of all machines. Extra information about [IP address](#) of the machine, user name and distribution is also displayed. The interface also allows for the machine to be powered off, turned back on or destroyed. The dashboard allows the user to download authentication certificate and gives instruction commands on how to access the machine.

This window makes it easy for a new user who has not used certificate based authentication to quickly access the machine. User experience techniques are also used to illustrate commands, failures, and important information. The dashboard is shown in figure 19. The page is also responsive, ensuring proper presentation across different screen resolutions.

Machines













Title	Distribution	Address	Action
temper-losing-divot-talkativeness (Logs)	ubuntu	user@88.168.56.253	   
<div>Status: Running</div> <div> <p>To securely connect to your machine, you have to use .pem certificate file to prove your identity. Download file now and store it securely, we will not store this file on our servers. If you download the file and lose it, that will make it impossible to connect to that machine.</p> <p>By clicking the link you agree that you have read the warning</p> <p>Download certificate</p> <p>To connect to this machine, navigate to the directory of the certificate and issue the following commands</p> <pre>chmod 400 temper-losing-divot-talkativeness.pem && ssh -i temper-losing-divot-talkativeness.pem user@88.168.56.253</pre> </div>			
oversaturating-fantastically-quirites-concreting (Logs)	ubuntu	user@88.168.56.252	   
<div>Status: Machine failed to deploy: Logs</div>			
rootage-twelve-crosswind-quirites (Logs)	ubuntu	user@88.168.56.251	   
<div>Status: Machine failed to deploy: Logs</div>			

Figure 19: Dashboard sample screen

8.9 Authentication and Credential management

The requirement Access is listed on page 4 in section ‘4.3.2’. The feedback from [NUIT](#) for using certificate files for authentication instead of passwords was positive. As part of the security concerns, this provides secure authentication channel. Future work should allow users to generate keys themselves and import them though the web interface, as well as revoke old certificate keys. The decision to give preconfigured authentication credentials is because of the target demographic. Potential early adopters of the platform will have little to no experience in the field and would appreciate the convenience of credential being already set-up.

8.10 Documentation

Currently little to no documentation is provided. This can be improved by writing articles on how to deploy the platform to a host with different configurations.

Also the inclusion of ruby documentation (rdoc) embedded in each source code file would improve code maintainability. This type of documentation is very helpful when new developers join to develop. Having a quick description of what each code function does can significantly reduce training time.

8.11 Firewall and package management

The requirements Firewall and Package install are listed on page 4 in section '4.3.2'. The solution provides the ability to open network ports to the outside world and to do package management during the install process via the web interface. Currently, few common packages can be installed to illustrate the functionality. Future work should go into finding the most vital packages and offer them as an option. This is possible due to the ability of the solution to extend packages, ports and deployment flow.

Something that is currently lacking is the ability to amend ports and packages. After the build process is over, one can manually connect to the machine and install/remove software but the process is not automated after the machine is already deployed via the web interface.

8.12 Package Mirroring

As part of the evaluation feedback, a suggestion was made to use package mirroring server in the future. This is a server that contains a [rsync](#) copy of the official distribution packages. The idea is to have everything necessary downloaded, making it installable to every new virtual instance almost immediately. This will have the effect of reducing bandwidth costs and reduce build times.

8.13 System resource validation

During the deployment process, a machine is validated on its own and the platform's state is not considered. For example, if there is not disk space available or the system is low on [RAM](#), this will not get detected and failure might happen. This has not been implemented because it requires managing virtual hard drives and storing meta-data about the virtual machine with status of the whole platform. It would require large sum of work and would take significant amount of work, possibly making it impossible to deliver the platform on time. Not tackling this issue helped allocate more time on the core components and deliver the assignment on time.

8.13.1 Security evaluation - Leaking passwords

When using the command line tool on the server platform, [UNIX](#) command `ps` can be used to see all arguments passed to the build tool. When using the `-password` flag, this will disclose it. For this exploit to happen, the platform owner must invoke the tool manually.

In the most common case, this command would leak the [API](#) key of a user. Future work might opt for using a hash of the [API](#) key and have the option of regenerating it. This would significantly reduce chances of further exploitation upon system breach.

8.13.2 Allocation of network addresses

A demonstration of [IP addresses](#) allocation was performed during the evaluation. Deciding to give network addresses incrementally received positive feedback. The reasoning behind using such approach is that no network conflict can happen and a stable address resolution can be achieved.

9 Conclusion

9.1 Fulfilment of requirements

The current solution successfully achieved the outlined objectives. It allows for secure access, management and configuration of a machine, guides the user through the user experience and makes it quick and easy to get started.

The platform creates a machine quickly and gives feedback to the user through the entire process and similar user experience has been provided throughout the solution.

9.2 Personal development

This project had a very positive impact on my professional development. I gained skills in the field of virtualisation and cloud computing. In the beginning, I had little to no knowledge on the subject, but now I am more comfortable and familiar with the concepts and methodologies. Building the system also helped me appreciate and adopt as part of my work flow automation and testing practices. Another benefit was practising development with different environments and managing features via version control.

Because the project was mainly concerned with the Linux [operating system](#), a large amount was learned about the inner workings of commands, using the shell to chain commands together (pipe) and fork processes.

In a world that adopts virtual machines more and more, it is also a valuable skill to understand the underlying platform, as opportunities to manage and maintain such system arise more and more.

Overall the work has helped me become a better professional with a diversified skill set, it has let me experience the process of delivering a solution from begin to finish.

9.3 Future work

Because of the limited duration of the dissertation and the work that has to be carried throughout the implementation (testing, documentation, development, debugging, planning, architecture, external tools, hardware constraints, etc.) certain limitations are put in place to ensure that the project will be completed within the deadline.

The initial constraints are defined to limit the solution to a single host. This work is done as a proof-of-concept system that has the potential to be scaled up and extended to support a clustered deployment. Creating a distributed and scalable implementation of this work goes beyond the scope of the objectives. This is not to say that it is limited to single host, but testing and planning guarantee that it will be fully functional under such condition.

It should be possible to deploy multiple instances of the underlying infrastructure managed and synchronised with a technology like puppet or chef. Then, creating virtual machines can be done via a manager machine which picks a host

based on a round-robin algorithm. Additional implementation steps must be carried out to ensure that the web platform and the manager understands the physical location of the virtual machine. Physical location references to the real hardware (the host) running the virtual machine.

The current solution makes assumptions about the available network interface. Right now, one must be connected to the network environment to communicate with the machines inside of the virtual network. VirtualBox's host-only networking interfaces is used to create multiple network instances that allow for the creation of machines independently of one another during developing and testing.

When further extending the solution, additional work has to be carried out to ensure integration with a public [DNS](#) server, and some adjustments ought to be made to allow the production [DNS](#) to allocate [DHCP IP address](#) to the virtual machines.

The solution aims to orchestrate and automate generic virtual machine deployment, but creating a solution that enables every [operating system](#) to be deployable would require extensive amount of additional changes. Future work should make it possible to deploy Windows, Mac OS and more Linux distributions as part of the infrastructure.

Windows is not supported yet, as it has completely different architecture and has a separate interface for managing networking, users, installation and permissions. Doing windows deployment also requires licensing of their [operating system](#). Another constrain in working with windows is that not many core components are configurable or accessible via a script making automation very difficult or impossible, or are not freely available.

Mac OSX virtual deployment was also something left for future. It also has similar issues to windows virtualisation (licensing, exposed [API](#)), as well as it is a platform that I have no experience with.

10 References

References

- [1] Antony Higginson Birgit Kreuz. *Migrating Oracle Databases*. Oracle Corporation, June 2014.
- [2] Buddy Burden. Perl vs shell scripts, April 16, 2012. Accessed: April 10, 2017.
- [3] Bill Calkins. *Oracle Solaris 11*. Prentice Hall, 2013.
- [4] Intel Corporation. Virtualization and cloud computing, 2013. Accessed: April 04, 2017.
- [5] Microsoft Corporation. Azure regions, 2016. Accessed: April 20, 2017.
- [6] Microsoft Corporation. The trusted cloud, 2017. Accessed: April 20, 2017.
- [7] Oracle corporation. OracleTMvm user’s guide, 2015. Accessed: March 08, 2017.
- [8] Nagios Enterprises. Nagios official website, 2017. Accessed: May 1, 2017.
- [9] Matt Goedtel et .al. Azure automation overview, October 5, 2010. Accessed: April 20, 2017.
- [10] National Technical Authority for Information Assurance. Virtualisation security guidance, May 19, 2016. Accessed: April 10, 2017.
- [11] Synergy Research Group. Amazon leads; microsoft, ibm & google chase; others trail. Accessed: April 02, 2017.
- [12] Hashicorp. Why vagrant? Accessed: 27.03.2017.
- [13] Garret Heaton. Amazon ec2 instance comparison. Accessed: April 04, 2017.
- [14] Charles Humble. Twitter shifting more code to jvm, citing performance and encapsulation as primary drivers, July 4, 2011. Accessed: May 1, 2017.
- [15] ITU ICT. Ict facts and figures 2016, 2015. Accessed: March 08, 2017.
- [16] Javvin Technologies Inc. *Network Protocols Handbook*. Javvin Technologies, 2005.
- [17] Techopedia Inc. Definition of virtualization. Accessed: November 26, 2016.
- [18] Lee Smith Jr. vsphere 5.5 storage profiles are now storage policies, October 28, 2013. Accessed: April 15, 2017.
- [19] Harrison Kinsley. Python programming tutorials. Accessed: April 03, 2017.

- [20] Puppet Labs. Puppet documentation. Accessed: 27.03.2017.
- [21] STEVE LOHR. Challenging microsoft with a new technology, August 30, 2009. Accessed: April 10, 2017.
- [22] Shannon Meier. *IBM Systems Virtualization: Servers, Storage, and Software*. IBM, April 2008.
- [23] Kate Miller. The case for investing in cloud automation, October 7, 2016.
- [24] San Murugesan. *Heterogeneous Cloud Computing: The Way Forward*. BRITE Professional Services, 2015.
- [25] Mike Perham. Sidekiq wiki home, March 1, 2017. Accessed: May 1, 2017.
- [26] Inc. Red Hat. Performance counters for linux (pcl) tools and perf. Accessed: April 05, 2017.
- [27] Margaret Rouse. Information technology (it), April, 2015. Accessed: April 05, 2017.
- [28] Gem Sawyer. Ruby guides, 2017. Accessed: April 13, 2017.
- [29] Core Security. Virtual pc hypervisor - memory protection, 17 March, 2010. Accessed: April 28, 2017.
- [30] Amazon Web Services. Scenarios for accessing a db instance in a vpc, September 27, 2015. Accessed: April 20, 2017.
- [31] Internet Live Stats. Google search statistics. Accessed: November 26, 2016.
- [32] Intel Support. Intel virtualization technology requirements, Feb 28, 2017. Accessed: April 10, 2017.
- [33] Jon Tate. *Introduction to Storage Area Networks*. IBM, 2016.
- [34] Ruby team. Ruby homepage. Accessed: 27.03.2017.
- [35] Ruby On Rails team. Getting started with rails. Accessed: 27.03.2017.
- [36] RubyGem Team. Make your own gem, 2011. Accessed: April 13, 2017.
- [37] Ryan Troy. *VMware cookbook*. Oreilly Media, 2012.
- [38] Inc. VMware. *Introduction to VMware vSphere*. VMWare Inc, 2009.
- [39] Vic (J.R.) Winkler. *Securing the Cloud*. Elsevier/Syngress, 2011.

11 Glossary

Glossary

Amazon Web Services Amazon’s cloud infrastructure as a service . 7

API Application Programmable interface, usually a way for exposing features of a certain program to the developer. Often used to automate a task. . 4, 11, 16

Azure Microsoft’s cloud infrastructure as a service . 7, 9

bash Bourne Again SHell, a UNIX command line virtual shell language. 10, 14

chef Tool for automating, deployment and configuring software. 7

CPU Central Processing Unit, the hardware component that is responsible for executing instructions, evaluate expressions and perform algebraic operations.. 6, 7, 9

DHCP Dynamic Host Configuration Protocol. 16

distribution Is a term that describes a Linux operating system with associated packages and pre-installed software . 13, 14

DNS Domain Name System, used to resolve internet addresses into human readable domain names. 8, 16

gem Ruby programming language term for module or package . 12

GPU Graphical Processing Unit, a computer hardware component that specialises in processing highly parallel instructions.. 8, 9

html Hypertext Markup Language, used for displaying web page . 25

I.T. or Internet Technology is defined by Techopedia as use of computers, storage networking and other devices and a variety of other hardware with the purpose of exchanging digital information [\[27\]](#) . 6

IP address An identifier that helps locating a resource on the network, usually represents a computer device connected to a network . 13, 14, 16, 25

kernel Is the core component of an operating system that provides all the services necessary for every component of the system.. 9

migration Migration is the ability to switch to a new version, platform, or physical location [1]. This term come from database management, but is also used in other computer science fields. . 2

NAT network A technique where one IP address is mapped to more internal ip addresses, used for security and reduced number of ips. [16] . 12

network mask A netmask is a 32 bits field whose p high order bits are set to 1 and the low order bits are set to 0. The number of high order bits set 1 indicates the length of the subnet identifier. Netmasks are usually represented in the same dotted decimal format as IPv4 addresses. [?] [p.143]netmask-definition . 13

open-source Software that makes its code openly available, allows derived work without restrictions and its distribution is done freely. 10

operating system The platform on which all user and system software is running. Examples are Windows, Linux, OSX, Android, iOSX. 1, 3–5, 9, 16

partition Bill Calkins defines partition as "Disks are divided into regions called disk slices or disk partitions. A slice is composed of a single range of contiguous blocks. It is a physical subset of the disk (except for slice 2, which represents the entire disk)." [3] . 2

puppet Open source Puppet helps you describe machine configurations in a declarative language, bring machines to a desired state, and keep them there through automation.[20] . 4, 12, 25

RAM Random Access Memory, also main memory, is a computer hardware component that stores information for fast retrieval.. 9, 14

ruby A dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.[34] . 12, 25

Ruby On Rails Rails is a web application development framework written in the Ruby language. It is designed to make programming web applications easier by making assumptions about what every developer needs to get started. It allows you to write less code while accomplishing more than many other languages and frameworks.[35] . 25

SSH SSH: secure shell, a network protocol that allows a remote connection to a terminal . 14

stand-up A daily team meeting where every team member gives an overview of done tasks, current tasks and any future work that needs to be done by that member. . 23

Vagrant Vagrant provides easy to configure, reproducible, and portable work environments built on top of industry-standard technology and controlled by a single consistent workflow to help maximize the productivity and flexibility of you and your team. [12] . 11–14, 25

virtual machine Oracle Corporation defines it "as a virtualized operating system with its associated software and applications" [7] . 7–9, 11

virtualbox Open-source virtualisation technology created by Oracle. 7, 10

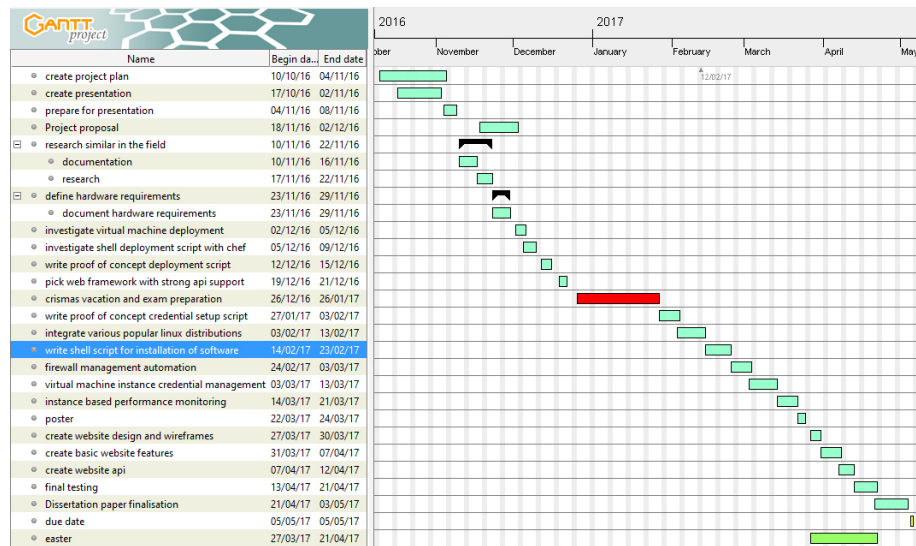
VmWare VMware is the leader in so-called virtual machine software, it is also the name of the product that the company produces [21] . 7, 10

vSphere Is a technology used to manage large collections of infrastructure (such as CPUs, storage, and networking) as a seamless and dynamic operating environment, and also manages the complexity of a datacenter[38] . 7

Xeon The Xeon processor family is Intel's server grade processors . 7

Appendices

A Planning



As part of the development cycle, I had a supervisor meeting every fortnight. The goal was to discuss work progress and ensure things are moving forward. During these meetings, I received feedback, raised concerns and showed progress in a [stand-up](#) fashion. Occasionally, the meetings were not about progress update, but about demonstration of the work done so far with the purpose of obtaining feedback and presentation practice.

A.1 First Semester

When creating the project plan, two core stages were taken into consideration. The first stage is semester one. During that period, the ground work was laid, on which all future development and testing was based. This was a critical moment into the building process. It allowed for a more in-depth research using materials created by the leading experts in the field (Amazon, Intel, Oracle). Their guides and documents helped gaining more understanding in various aspects of building cloud-ready applications, infrastructure security and automation.

The research helped when rationing about the features that need implementing. It also illustrated the scope, and the challenges that need to be faced. The research gave an insight into why this problem requires solving, and pointed me to open-source projects that achieve parts of the work.

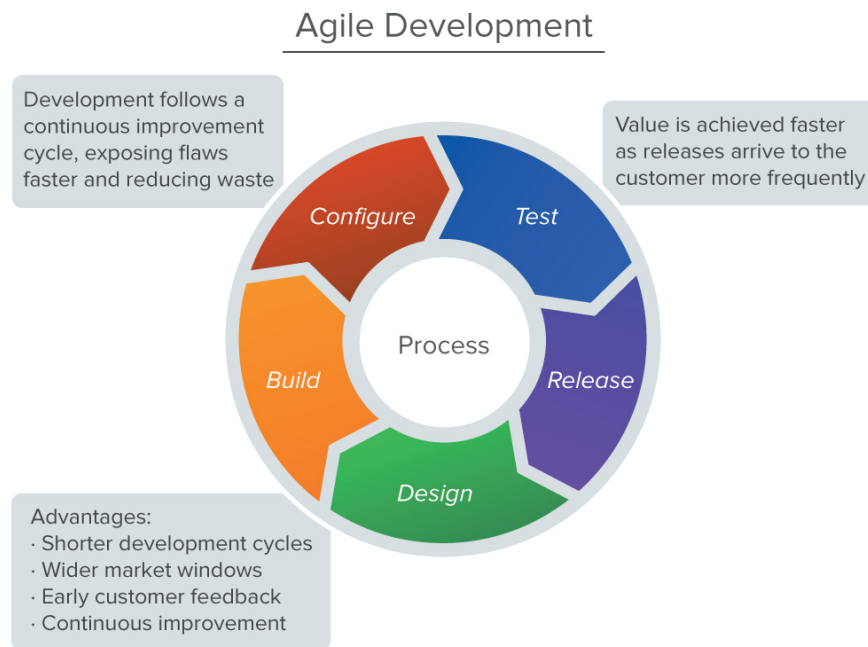
The first semester was not only about research, but also about describing the constraints, limitations and infrastructure decisions that need to be made in order to demonstrate a product that has real-world applications. The materials

helped me also to identify potential risks and issues that might arise, such as the security risk of running multiple virtual machines onto the same physical space. This is a potential risk situation, as an attacker can read the private data of another machine if the person gets access to the memory of the "host".

Work also went into creating presentation and project proposal for the upcoming work, which gave an outline and highlighted the features that the work addresses. When creating the presentation, I wanted to be as generic as possible as not to present information and ideas that the additional research might prove impossible to implement or not practical.

A.2 Second Semester

The tasks accomplished during the second semester are related to implementing, refining, iterating, evaluating and documenting the solution. This is the time where all the main features, functional and non-functional requirements got developed, tested and documented. It must be noted that for the most part, each cycle was not incrementally tackled, but instead, numerous development-testing-documenting phases were used to achieve the final result.



For the most part, the project was completed using agile principles. Each cycle that shaped the final product is described below.

A.2.1 Development

As planned, the work was broke down to subtasks. The initial plan is not to develop the whole system and assemble the parts together at the end, but to build a minimum viable product (MVP) and increment and improve onwards.

This was done in order to assure that the platform can be built with the available tools and that the requirements can be met at the end of the project.

Being able to create a virtual machine with a known [IP address](#) from a script was the first thing that had to be accomplished. Creating custom users, opening ports and creating the web interface was planned later as something that could be built on top of the "core" product.

A.2.2 Learning period

For the duration of this project, a variety of new technologies were tried and evaluated, such as [Vagrant](#), [puppet](#), [Ruby On Rails](#), [ruby](#).

As part of the learning cycle, the official ruby programming language manual was used. It helped gain proficiency and to understand the fundamental constructs that were used throughout the tutorials, articles, blogs and other learning materials. The process taught me about declaring variables, scopes, for loops and any extra syntax unlike other programming languages. To get up and running with [ruby](#), the period between first and second semester was planned to build basic command line programs.

Another part of the development strategy was to do a small web project using ruby on rails just to get a feeling of what is possible in terms of rapid development and ease of use. This small project included building forms, validation, manipulating database objects and creating [html](#) pages. This was done in February (the beginning of second semester), with the purpose of determining if this tool is right for the job.

The task of learning [puppet](#) was achieved by creating end-to-end automation scrip of an apache server doing various tasks. Puppet's power is to get a machine's configuration to a known state and ensure all software packages are present. Doing this exercise was overall a good practice, as it led to a faster implementation and integration later on when working on the actual platform.

A.2.3 Documentation

The initial plan as stated in the project proposal is to document the work during each stage of the process, like implementing a new feature. Later, the decision was made to not strictly follow that plan, as there were few components that changed drastically. For example, in the beginning, using chef was planned for configuration automation and management instead of puppet. After evaluating and experimenting with it for 5 days, it was apparent that the set-up overhead was slowing down the actual development process.

Because of that experience, the dissertation write-up process was changed to documenting features only when they are established. And reasonably stable. The document paper is written using the latex language.

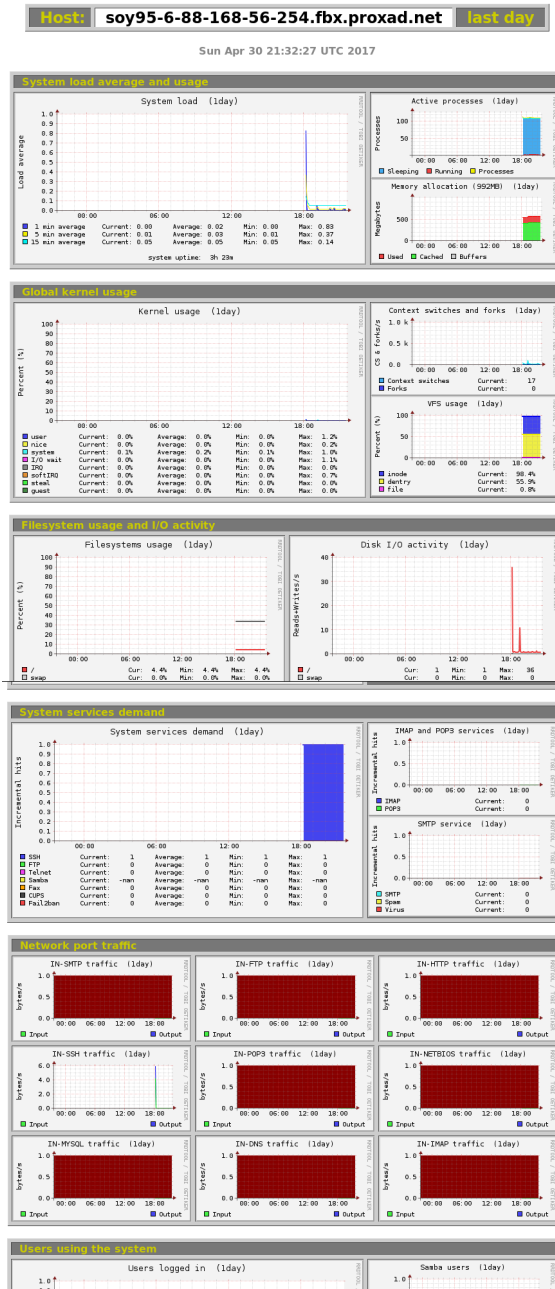


Figure 16: Monitorix view from the web interface