Google Cloud Platform

Friday 10th May, 2019 - 00:42

Kevin L. Biewesch
University of Luxembourg
Email: kevin.biewesch.001@student.uni.lu

Alfredo Capozucca
University of Luxembourg
Email: alfredo.capozucca@uni.lu

Abstract

This BSP is essentially a continuation on the last BSP which tackled the issue of easing the creation of new environments. While last semester we worked locally on our machines, meaning that we created these environments on our pyhsical machine, this semester we work in the cloud, so the environments will be hosted on a remote server which we can access. Thus we are taking advantage of cloud computing.

We are reusing part of the work that has already been done, mainly the scripts for provisioning and the assessment, and will expand upon it to fit the wider scale and diffrent nature of this project.

1. Introduction (± 5% total words)

Over the course of this BSP, we are trying to achieve an automated solution that eases the creation and setup of a new environment, more specifically the Excalibur Environment.

To achieve our solution, we worked with the Google Cloud Platform (GCP) to create our instances. Later on we found ways to automate the creation of new instances on GCP.

In order to have all the tools and packages needed for the Excalibur Environment as well as for ensuring remote desktop access we used *Ansible* which allowed us to provision our instances. That way, all the required stuff will be put on the machine automatically.

While trying to provision our machines however, we encountered an issue that required us to look into how to set up SSH connections.

Finally, we also produced a lengthy tutorial that explains all the things you need to know. Topics range from setting up GCP all the way to some insight on the tools used during the BSP.

2. Project description (± 10% total words)

2.1. Domains

There are two big domains that are being tackled in this BSP. On the one hand we have virtualization and on the other hand we have cloud computing along side its service models.

2.1.1. Scientific. Cloud computing brings the domain of virtualization, as explained in the next section, to yet another level. We are still manipulating a virtual medium, but the virtual machine does not run locally on our physical machines. Instead these machines run on some servers and we can controll them through a remote connection. This leads us to the various service models employed by the cloud.

The different service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The three service models are briefly explained in the following.

- **IaaS** We are provided with the necessary infrastructure to run a virtual machine and the rest is up to us. This is the model used by Google Cloud Platform.
- **PaaS** We get the necessary resource as well as an OS installed with some basic software. We can use the provided platform and modify it as we please. However we are not able to touch the underlying virtual hardware.
- **SaaS** Usually SaaS is accessed through a web browser and we are provided with everything needed to run a specific software. We are only able to access the software and none of its underlying components, like the OS or the hardware.

2.1.2. Technical. Virtualization allows us to move from physical mediums to virtual ones without really losing anything. The only major difference is that now we do not manipulate the hardware directly but instead manipulate the virtual hardware through specific software. This allows for great flexibility in a lot of ways.

Along with virtualization, we dive into a concept called *Infrastructure as Code* which essentially means that we can describe infrastructure[1] through code or configuration files. For example, we can write a little piece of code that creates a virtual machine and inside that code we can give specifications for the machine, like disk and memory size.

Furthermore, the concept of *provisioning* will come into play when talking about Infrastructure as Code. Provisioning allows us to further specify characteristics of a machine. With this, we can have various tasks be performed automatically. When calling this provisioning while creating the virtual machine, we can for example specify packages that should be installed right away or other operations that should be made to obtain the machine we want.

1. Infrastructure refers to the hardware and software that compose a machine

Lastly, the concept of provisioning brings us to a tool called *Ansible*. This is a tool specifically designed for provisioning machines with great ease. With Ansible, we can write playbooks that basically say what should be done in order to provision the machine. The language for these playbooks is *yaml*, which is very easy to use. This way we end up with a bunch of *configuration files* that describe that machine.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. Firstly, we had to get familiar with GCP since this is the tool we used to create our virtual machines[2]. This means that we had to learn and understand how to work with the platform.

While trying to achieve the provisioning on the instances, we encountered a problem that required us to familiarise with private and public key pairs to get remote access to GCP instances.

Lastly we want to create a tutorial that accompanies our technical solution. The tutorial is meant to guide the DevOps engineer and help him understand and use our solution.

2.2.2. Technical deliverables. We have to consider two types of users: the DevOps engineer and the Excalibur user.

As for the DevOps engineer, the objective of this BSP is to have a fully automated solution that handles the virtual machine creation on the Google Cloud Platform. Added to this, all the necessary tools should be downloaded such that we can put and Excalibur Environment on it. He also needs to ensure access to the new virtual machine for the Excalibur user.

As for the Excalibur user, we want him to simply be able to connect to the virtual machine created in the cloud and use the Excalibur Environment. The end user should not worry about anything related to the virtual machine's creation and setup.

3. Background (± 10% total words)

3.1. Scientific background

Virtual machines constitute the big scientific aspect to be familiar with for this BSP. They were already briefly explained before in section 2.1.

Also the concept of provisioning should not be foreign. In essence, provisioning allows us to put files and packages as well as execute scripts onto a virtual machine without accessing it directly. We can write scripts, or in our case configuration files, that will then access the machine and accomplish all the specified tasks.

3.2. Technical background

The technical aspect to be familiar with for this BSP were how provisioning works with Ansible. This is important because we are trying to reuse the work done in the previous BSP and get it to work on our virtual machines in the cloud.

Ansible is a tool that allows to write configuration files that basically specify what a machine should be provisioned with. It allows for great flexibility and keeps the focus on easy to read and write code.

4. Scientific Deliverable 1 – GCP

4.1. Requirements (± 15% of section's words)

The first scientific deliverable for this BSP concerns the Google Cloud Platform. In order to be able to work on this project, we had to understand how to work with the platform in order to ease the set up of virtual machines, as we will rely on this tool for the final solution.

4.2. Design (± 30% of section's words)

In order to learn about GCP and understand how it works, we decided to look for online courses and tutorials. These courses were meant to teach us the basic usage of the platform as well as a few basic concepts related to it.

The first tutorial [1] we watched was a single long video that briefly explained all the concepts related to cloud computing, thus giving a first overview of what we will be dealing with.

After that, we watched two tutorial series [2] [3], although not in their entirety, created by Google themselves that gave a more detailed insight into the various concepts and how they work in practice through a set of exercises.

Finally, when first accessing GCP a message popped up proposing a guided tour through the platform. This was basically an interactive tutorial that showed us how to navigate the platform and where to find the most important tools.

4.3. Production (± 40% of section's words)

The tutorials allowed us to understand the core concepts of the platform and how they relate to each other. This understanding is materialised in the glossary shown next.

1) **Projects** GCP projects form the basis for creating, enabling, and using all GCP services including managing APIs, enabling billing, adding and removing collaborators, and managing permissions for GCP resources. [5] For our sole purpose of creating remote virtual machines, we needed to create a project in which to create those. Added to this, we can set project-wide settings that will affect a lot of things, including virtual machines, that are contained within it. For example, we have a project in which we set a *default user* that should be used for SSH connections. That way we do not need to specify explicitly it for every instance. So you can consider projects as a sort of working environment.
2) **Instance** is a virtual machine hosted on Google's infrastructure. [6]

2. On GCP they are referred to as *instances*

3) **Billing** is an important part of GCP because the services are not provided for free. Contrary to a lot of services out there, we do not have to pay a fixed monthly subscription fee to use them. The way it works for cloud computing in general is that you pay for what you use. What this means is that if you have a bunch of instances, which obviously require resources to run, and they are *not* running, or in other words: they are shut down, then you do not pay for them. If you have instances that are running, then the amount you pay mainly depends on the resources the instance uses. You may even get discounts in specific cases.
4) **IaaS** is an instant computing infrastructure, provisioned and managed over the internet. It's one of the four types of cloud services, along with SaaS, PaaS, and serverless. [7] We did not touch on the the last one.
This is a concept that has been touched upon in the tutorials along side the other service models. As we have seen before, these are part of cloud computing. IaaS is important here because using GCP means that we rely on such a service. Essentially, Google provides us with the hardware and some software that comes with a chosen OS, so in other words the infrastructure, which we get to decide how to use.

4.4. Assessment (± 15% of section's words)

To assess this deliverable, we basically check if we were able to use GCP. As a matter of fact, we were successful in creating virtual machines and in manipulating them so we can consider this deliverable as successfully executed.

Furthermore, some of the relevant concepts have been explained in the tutorial, which should further highlight the understanding of the platform.

5. Scientific Deliverable 2 – Private/Public Keys for SSH access

5.1. Requirements (± 15% of section's words)

After having accomplished the creation of instances on GCP, we wanted to provision them using Ansible and the work we have done in the previous semester. We quickly encountered an issue where the Ansible script was not able to connect to the machine and perform the provisioning. As a result we needed to find out how to access GCP instances remotely via SSH.

5.2. Design (± 30% of section's words)

To learn how to get SSH access to GCP instances we had to options which we both exploited and combined.

The first option we had at our disposal were the guidelines provided by GCP. The platform provides a lot of documentation on a lot of topics and we managed to find some help about *Connecting to instances* [4]. It explained various ways to connect to various types of machines. However, for our purpose of provisioning SSH connections seemed like the way to go.

To further deepen our understanding on how to manage SSH access we also read blogs [9] [10] and tutorials [8] that were rather unrelated to GCP. Since SSH is not a specific feature of GCP we were, in the end, able to apply what we have learnt.

5.3. Production (± 40% of section's words)

SSH connections rely on so called *key pairs*. This key pair is composed of a private and a public key. To create this key pair we simply had to run the following command `ssh-keygen`. We will then be prompted to name the file and to give a password with which to encrypt them. If we do not keep the default name then we have to specify where the private key is each time we want to connect through SSH. If we set a password, we will be asked to enter it each time we want to connect to an instance using the password protected key.

Once we have the keys, we are half way there. We will notice that two keys were generated. One of them is the private and the other one is the public key. The latter one has `.pub` attached to the end. The last step is to put the public key onto the remote machine inside its `$HOME/.ssh` folder and we are good to go. GCP also has a feature that allows you to embed the public keys within the project. That way you do not need to put them manually on the instances.

We should note that the private key is not meant to be shared. To understand how it works, you can think of it as a real physical key you use to unlock the door to your house. In this scenario the public key is the door lock. You would not want to share your key with everyone because you might end up having unwanted guests at your place. The same applies to the private key. Anyone with your private key can access machines on which you have put your corresponding public key. We want to avoid this as it can lead to security issues.

5.4. Assessment (± 15% of section's words)

To assess this deliverable we simply had to verify whether or not we were successfull in accessing our instances using SSH and thereafter if we were able to provision them using our Ansible scripts. As a matter of fact, after setting everything up for the SSH connection we were able to connect and to provision our instances with our Ansible scripts.

6. Scientific Deliverable 3 – Tutorial

6.1. Requirements (± 15% of section's words)

The final scientific deliverable for this project is a tutorial targeted towards a DevOps engineer and is meant to help him understand and use our solution. The tutorial should talk about everything that you may need to understand how the solution works and to build upon it independently.

The functional requirement for this tutorial is, as already said, to help the DevOps engineer in fulfilling the main techincal solution of this BSP, namely to get a new Excalibur Environment.

The non-functional requirements include:

1) **Completeness**, which specifies how well the tutorial explains the given facts. To phrase it in a negated manner: will the reader still have major doubts after reading the tutorial?
2) **Easy to read**, determines the reading quality of the tutorial. We want the tutorial to feel light which in turn allows for an easy reading experience.

6.2. Design (± 30% of section's words)

Since the very beginning of the BSP, we were thinking about the best way to present the tutorial and convey its content. There were basically two options we were considering.

1) **Video format**. This would not be a bad decision because it is easy to follow. The person making the tutorial can precisely show how to do the various steps and what to click and so on. It would also be very easy to highlight potential issues and side information on the fly. In textual form if not done well, presenting such information can quickly make the tutorial lose focus from the topic at hand. Plus, there is a video presentation to do for the BSP, so that would allow me to practice recording a video and reuse some of the work done.
   However, making these videos takes quite a lot of time and preparation. In case of mistakes you would have to record a specific section again and maybe even multiple times if you repeatedly make mistakes. If you notice a problem a little further down the line, it might also be hard to fix.
2) **Textual format**. The main advantage here is that it is a lot faster to produce, since mistakes can be fixed with more ease and less effort. You can also rearrange and restructure the text without losing continuity because if you use transitions[3], rearranging parts of the video may make it lose continuity while in textual form, you simply remove or change these transitions.
   In case you already know how to use the solution or have read the tutorial and you have some doubts later down the road, you can quickly refer to a specific section from the tutorial. In addtion, if it is well structured, the tutorial allows for a quick and easy read and you can provide things to be aware of and miscellaneous information without deviating from the main topic.
   Nonetheless, it is a little harder to visualize steps to be done. If we include images, for illustration purposes, that are not annotated or described well enough, they might lead to confusion[4].

3. i.e. saying "elaborated in the next part","as previously mentioned", ..."
4. i.e. not understand where they belong or what they are meant to illustrate

In both cases, you may also run the risk of not structuring your tutorial well which makes it hard to read or watch and lose focus of the main topic at hand.

Considering all this, we opted for a tutorial in textual form, mainly due to its flexibility and faster production.

6.3. Production (± 40% of section's words)

The tutorial has been written in Markdown. We decided to go with Markdown because it is quite easy to use and you can add it to a GitHub repository. Considering that we will put this solution onto GitHub[5], this seemed like a very good idea. This way the tutorial is included with the solution and is presented nicely. Find more technical information in the appendix section 9.2.

To begin with, we needed a way to write Markdown files. I used to write my Markdown files with ReText[6] however I had a few issues with it and the output did not look all that nice. I tried switching to Vim, which has a pluggin for supporting real-time Markdown rendering in a browser[7] but again I had an issue here and it did not even seem to work. I kept searching a bit more and I found Remarkable[8] which was pretty good for the purposes of writing the tutorial.

Once we were set for writing the tutorial, we began with writing down things we already knew how to do. So we build the tutorial as we went, feeding it with new information as we learnt and discovered it.

Some parts of the tutorial required us to completely restart from scratch because once you have everything set up, there are things that differ compared to doing it the first time around. Especially all the things related to GCP have been done from scratch to show to the user everything he needs to know to get started. Once everything is set up there are fewer things to worry about, but they are important to highlight the first time around.

Markdown has the ability to insert images, so we used it to our advantage to illustrate a lot of things. See an example of a picture in figure 1. Furthermore, it allows us to include references, either external or internal ones. So we can have references to other websites or to other sections within the file.

We also used this to our advantage, to introduce various other notions, without interrupting the flow of the text. See figure 1.

Here we have, for instance, the table of contents which is followed by a few items colored in blue. This means that you can click on it to jump some place else. For the table of content, it makes you jump to the corresponding section in the tutorial. Also, in the first bullet point in the figure, you can see that *GCP* is colored blue. This one will actually take you to the Google Cloud Platform. We used this all throughout

5. See appendix section 9.1
6. https://github.com/retext-project/retext
7. https://github.com/suan/vim-instant-markdown
8. https://remarkableapp.github.io/

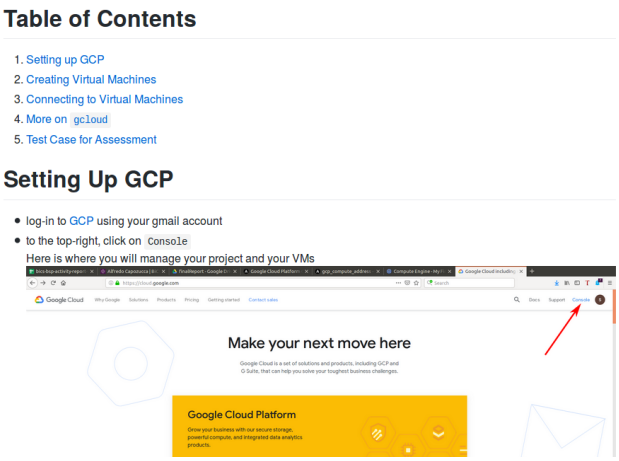

Fig. 1. Markdown example

the tutorial to send the reader, in case of need, to a specific section to get a refresher or some additional information.

6.4. Assessment (± 15% of section's words)

The assessment of the tutorial is made by providing it to a list of persons given by the BSP tutor. These persons will respond a survey. The answers of such a tutorial are used to determine the quality of the tutorial in terms of suitability.

7. Technical Deliverable 1 – Provisioning

7.1. Requirements (± 15% of section's words)

The main techical deliverable for this BSP, is a solution that automates the process of creating and provisioning virtual machines on the Google Cloud Platform in order to host an Excalibur Environment on it. Part of this solution is based on earlier results that have been worked out in the previous semester.

However, due to time constraints we were not able to fully realise this objective. So we left out the very last part where you actually have to install and set up the Excalibur Environment. We know for a fact that this should not be an issue thanks to the work done in the previous semester.

The functional requirement of this solution is to set up a new Excalibur Environment as fast as possible while needing the least amount of effort. We want as much of the set up as possible to be done by simply executing the provided scripts.

The non-functional requirements are more numerous. We want to tackle:

1) **Operability** [11], which determines how accessible and easy to use the solution is.
2) **Efficiency** [11], which considers the amount of effort necessary to bring the solution to use.
3) **Satisfaction** [11], informs us on the degree to which user needs are satisfied when a product or system is used in a specified context of use.
4) **Maintainability** [11], considers the ease at which the developer can change aspects of the Virtual Machine creation and the provisioning.
5) **Reliability** [11], tells us whether or not our solution is reliable. In other words: Does it give the same result in successive trials?
6) **Adaptability** [11], considers the degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments.
7) **Performance efficiency** [11], considers the the performance relative to the amount of resources used under stated conditions to evaluate this non-functional requirement.

This technical deliverable is targeted towards a DevOps engineer, who is in charge of setting up these machines. The Excalibur user should not have to deal with anything regarding the production of the final product, namely the Excalibur Environment.

7.2. Design (± 30% of section's words)

As already mentioned, part of this deliverable relies on work done in the previous semester. In the last semester, we also tried to produce an automated solution, however we were working locally on our machines. In this semester we tried to adapt the knowledge aquired and bring it to the next level, namely cloud computing.

While the basic concepts regarding virtualization remain the same, we now have to look at the subject from a different perspective which is due to the different nature of working locally versus working in the cloud. Before, all the DevOps engineer could do was to provide scripts the user had to run himself. We considered there was no way for a DevOps to interact with the user's machine which means that actually creating the machine with the provided solution was up to the user.

Now we want a DevOps to create and set up the machines in the cloud and the Excalibur user will merely have to connect to it and is ready to go. There is close to nothing that the user will have to do in order to get access to an Excalibur Environment. Essentially, what we are trying to achieve is Software as a Service.

A first difficulty for this BSP was to create virtual machines in the cloud. We had to get accustomed to the Google Cloud Platform and learn how to use it. This was a relatively easy and straight forward process because Google provided interactive tutorials to guide its users through the basics of the platform.

After getting the hang on the basics, we immediately started wondering if we could reuse the scripts that handled the provisioning from the previous semester. We opted for a provisioning solution that was not specific to the previous semester's solution but would allow for a broader range of application. A quick research revealed that indeed, we were able to reuse the provisioning scripts. Only one minor

modification had to be done, namely *how* do we execute it[9].

Lastly, we inspected the `gcloud` command line tool which basically allows us to interact with GCP through the terminal. We wrote scripts around this tool to have new machines be created automatically. Automating the provising also relies on some output generated by this tool.

7.3. Production (± 40% of section's words)

The first thing we had to get comfortable with is the Google Cloud Platform. There were a few things we needed to know and had to set up before we could even get started with creating virtual machines.

A first requirement was to have a gmail account to be able to operate on GCP. Next, we need to create a new project in which we manage and manipulate our virtual machines. You can read about it in the tutorial on github[10]. To see the steps necessary to achieve this, please refer to the tutorial[11].

After having created a first virtual machine[12], with very few resources attributed to it because it was merely a test and we did not want to have high billing for this, we inspected ways to connect to it[13].

The first obvious choice was to connect directly from GCP. However, this did not allow us to provision the machine using our Ansible scripts. So we were forced to inspect manual SSH connection. We had to ensure SSH connection to the virtual machine in order to run our Ansible provisioning.

This is due to the fact, that Ansible relies on a host file or inventory file as it is called. This file contains a list of hosts to provision and Ansible will connect to these host via SSH. There seem to be methods for connecting to a host on GCP from Ansible without requiring manual SSH connection, but these methods looked rather complex and we did not get them to work. That said, we stuck to manually ensuring SSH connection.

```
[GCP]
35.227.50.0

[GCP:vars]
ansible_user = student002
```

Here is an example of a host file that was written manually in order to check if provisioning works. With the `[...]` we can specify a group of hosts. That way we can have Ansible provision only a specific collection of machines if we so desire. However, in the Ansible scripts, we have set it such that all hosts will be provisioned.

Furthermore, we have the `[...:vars]` with which we can specify variables that apply to a certain group of hosts. Here we say what user to connect as on the remote machine. With the scripts that automate the solution, our host file will be generated and look a little different.

```
35.227.50.0  ansible_user = student002
35.653.43.0  ansible_user = other_user
```

This way we are specifying the *main* user for each machine. Since each machine will have multiple users we need to specify a user, with which to connect as, to do the provisioning. The multiple users are required for ensuring Remote Desktop Protocol (RDP) connection. Somehow it is not possible to connect to the remote machine's desktop with the main user, so that is why there will be multiple users.

After being able to handle the provisioning, which is a big step in the right direction, we looked into how to automate the creation of the virtual machine. Our first attempt, was to integrate this into our Ansible workflow. There are methods provided by Ansible to create virtual machines on GCP but here again, we did not manage to get them to work. So we went for the gcloud tool for doing this. The gcloud command we used to create virtual machines is a follows. More details on the command can be found in the tutorial[14].

```
gcloud compute instances create "$1" \
    --machine-type="$MACHINE_TYPE" \
    --boot-disk-size="$BOOT_DISK_SIZE" \
    --image-project="$IMAGE_PROJECT" \
    --image-family="$IMAGE_FAMILY" \
    --metadata="$METADATA"
```

After having all of this figured out, we wrote one script around the above command to handle the virtual machine creation and another script is used to handle the provisioning. This second script relies on gcloud to get the external IP of the remote machine because the IP changes each time you start it anew. So we write the IP along with the *main user*, as explained above, to a new host file and have ansible provision the machines specified in that host file.

To wrap things up, we wrote a third script that calls these two scripts. That way we only need to execute one script and everything is handled for us. The last script has the following content.

```
#!/bin/bash

# $1 : instance name
# $2 : main user name at the instance

bash ./new_vm.sh "$1"
bash ./provision.sh "$1" "$2"
```

7.4. Assessment (± 15% of section's words)

We will evaluate the solution with regards to the non-functional requirements to check how well these have been tackled. Admittedly, the only part of the final solution that has not been solved yet is to automatically ensure SSH and

9. In the previous semester, the executing of these scripts was very specific to the solution
10. See appendix section 9.1
11. Section **Setting up GCP**
12. Section **Creating Virtual Machines**, subsection **Manual Creation**
13. Tutorial section **Connecting to Virtual Machines** and its subsections
14. Section **Creating Virtual Machines**, subsection **Creation using gcloud**

RDP[15] access once the instance is created. So this a step which the DevOps engineer needs to do manually before being able to respectively provision the machine and set up the Excalibur Environment.

Also, we will consider that the DevOps engineer uses this solution more than once since the first time around there are quite a few things to set up but after that you are good to go and can deploy new instances quite fast. So for the evaluation we will not consider the things you have to do only the very first time.

We will use the following scale ranging from handled the worst to best for evaluation purposes: −−, −, +, ++.

1) **Operability**. In order to evaluate this we will take into consideration all the things the DevOps engineer must be able to do beforehand in order to tackle the given solution.
   For our solution, the DevOps engineer must at least know how to set up GCP and a project, how to set up the gcloud command line tool and how to get SSH access to the instances. After that, the scripts handle pretty much everything. Granted, the tutorial we provided gives step by step instructions for these tasks. Thus we conclude that the operability is +.
2) **Efficiency**. In order to evaluate this non-functional requirement, we will look at the amount of steps necessary to acquire the Virtual Machine and get them ready for the Excalibur Environment. We will orient ourselves on the tutorial that will be presented along side the script.
   If you want to create the virtual machine manually it will take you, give or take, 3 steps. If you opt for the script or gcloud tool there is only 1 step. Although the gcloud tool might take you a few more steps to find the right things to put in the flags.
   For provisioning you have to set up SSH access manually and afterwards you need to set up RDP access as well. These two steps might be a little cumbersome. So the efficiency is +.
3) **Satisfaction**. In order to evaluate this, we will again look at the amount of steps necessary but also at the time required to execute them. This is due to the fact that more and lengthy steps to follow decrease the ease and speed at which you can repeat a given solution.
   We have two steps on SSH and RDP that need to be done manually and may that may be a little annoying, but at the same time we give the user the possibility to use an Excalibur Environment with great ease. In fact, the Excalibur user does not have to do anything at all. Despite the two manual interactions we will say, thanks to the great satisfaction on the user's side, that satisfaction is ++.
4) **Maintainability**. For evaluation of this non-functional requirement, we look at how difficult and how much effort it may take for the developer to to tweak the creation and provisioning.
   The two steps that require manualy intervention, namely the SSH and RDP setup, do not give rise to maintainability issues because you set them up once and there is nothing to worry about anymore. Added to the fact that everything is handled by scripts which you can go and edit, we conclude that maintainability is ++.
5) **Reliability**. For our evaluation purpose, we will try to look at how many things can go wrong during the Virtual Machine creation. So the fewer things there are that can go wrong, the more reliable the solution will be.
   Except for the two steps in setting up SSH and RDP nothing can really go wrong because the script handles the rest. For setting up SSH there are not too many things that can go wrong considering there are not a lot of steps and they are rather simple. However the RDP setup requires a few steps that may or may not demand a little more attention, since you need to have a look at one or the other configuration file. Hence we will say that reliability is +.
6) **Adaptability**. To evaluate this non-functional requirement, we will consider the ease with which you can adapt an instance to a given situation.
   Due to the fact that instances are not physical, we can pretty easily tweak its properties. This means that we can give it more resources if the current ones are not sufficient[16]. We should note that this is not as easily accomplished with physical machines as with virtual machines. Hence adaptability is ++.
7) **Performance efficiency**. For evaluation purpose, we will look at how good of a performance we can achieve with a given set of resources.
   Thanks to the fact that we can tweak the instance's properties, like resources, we have the ability to try and fiddle around with the amount of resources until we find a specification that yields the greatest performance with the least amount of resources. Due to the potential fiddling, we will say that performance efficiency is +.

Non-Functional Requirement	Evaluation
Operability	+
Efficiency	+
Satisfaction	++
Maintainability	++
Reliability	+
Adaptability	++
Performance efficiency	+

TABLE 1. NFR summary table

The summary table 1 gives an overview of this non-functional requirement evaluation.

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done.

15. Remote Desktop Protocol is used for sharing the Desktop of the instance

16. We are assuming that the servers where our instances are running on have enough resources to allow us to do this

8. Conclusion

The conclusion goes here.

References

[1] Introduction to Cloud Computing. https://bics.udemy.com/introduction-to-cloud-computing/learn/v4/overview
[2] Google Cloud Platform Fundamentals. https://www.coursera.org/learn/gcp-fundamentals/home/welcome
[3] Google Cloud Platform Infrastructure Foundation. https://www.coursera.org/learn/gcp-infrastructure-foundation/home/welcome
[4] Connecting to instances. https://cloud.google.com/compute/docs/instances/connecting-to-instance
[5] Creating and Managing Projects. https://cloud.google.com/resource-manager/docs/creating-managing-projects
[6] Virtual Machine Instances https://cloud.google.com/compute/docs/instances/
[7] What is IaaS? https://azure.microsoft.com/en-us/overview/what-is-IaaS/
[8] Configure SSH key based authentication. https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server
[9] SSH Key. https://www.ssh.com/ssh/key/
[10] Set up SSH public-key authentication. https://kb.iu.edu/d/aews
[11] ISO/IEC 25010:2011. System and Software Quality Models

9. Appendix

9.1. Final solution on GitHub

The final solution along with the tutorial as well as this report are on github: https://github.com/niveK77pur/ExcaliburEnvironmentGCP

9.2. More on Markdown

9.2.1. Images. The command we used to insert images in the Markdown file is as follows.

```
<img src=Images/IMAGE_NAME.png width=1000>
```

We used this html version instead of the default `![](Images/IMAGE_NAME.png)` because here you cannot specify a size for the image. In Remarkable, the images were way too big and it became quite unreadable, so we went with the html version since it allows us to scale it to 1000 pixles in width.

9.2.2. References. Internal references need a tag to be referenced. The tag and the reference look like this in Markdown.

```
<a name=text-tag></a>
Some text with a tag at the beginning.

This is an [internal reference](#text-tag)
while this is an
[external reference](https://www.google.com)
```