

Proof-of-concept for an Infrastructure as Code Teaching Module

Dominic Routley - C1534969

10th May, 2019

CM2303 - Final year project
School of Computer Science and Informatics, Cardiff University



Project supervisor

Project moderator

Padraig Corcoran
`corcoranp@cardiff.ac.uk`
David Walker
`walkerdw@cardiff.ac.uk`

Dedicated to Faith Routley,
for everything.

Abstract

This report describes and explains a series of resources that have been created for use in a university teaching module based around Infrastructure as Code (IaC). These resources make up a proof-of-concept for a finished module section or small module. The report then goes on to lay out the future work that would be required to turn the proof-of-concept into a fully finished module section.

Acknowledgements

I would like to thank the following people for their invaluable support with this project.

Padraig Corcoran `corcoranp@cardiff.ac.uk` Project supervisor
For support with everything and making sure I actually produced something useful.

James Osborne `osbornej8@cardiff.ac.uk` Placement supervisor
For his encouragement during and after my internship that this was a viable project idea. Also for his help understanding the NSA.

Daniel Rees `daniel.rees@devopsgroup.com` Industry specialist
For his professional advice regarding code examples, and pointing out code mistakes in my lecture slides.

Jamaal Primus `jamaal.primus@devopsgroup.com` Industry specialist
Technical proof reading (coursework deliverable)

Colin Barker `colin.barker@devopsgroup.com` Industry specialist
Technical proof reading (coursework deliverable)

Faith Routley
Proof reading (report) & Personal support

Henry Routley
Proof reading (report)

Nancy Routley
Proof reading (report)

Ryan Cullen
Personal support

Lucy Young
Personal support

Contents

	Page
Contents	iii
0 Introduction	1
1 Background	3
2 Deliverables	5
3 Future work	14
4 Summary	19
5 Reflection	21
Appendices	23
A Coursework Pro forma	23
B Lab session handout	28
Glossary	31
Acronyms	35
Bibliography	36

Chapter 0

Introduction

This project is intended to facilitate the creation of a proof-of-concept university teaching module that focuses on cloud engineering and cloud operations. The section that has been decided upon for prototyping and exploring in the proof-of-concept phase was to be focused around IaC.

The Computer Science BSc degree at Cardiff University does not include a teaching module (either optional or core), that explores cloud computing in detail. As cloud engineering and operations have quickly become a large part of the information technology industry, this project will design and partially implement a proof-of-concept module that could be used as a basis for creating a fully fleshed out module in the future. There is currently a MSc teaching module entitled ‘Distributed and Cloud Computing’ for which this teaching module could serve as its BSc introduction to prepare students for the MSc course. Distributed systems is already taught to BSc students in the third year as part of the ‘High Performance Computing’ and ‘Emerging Technologies’ teaching modules. A cloud operations and engineering module will complement the third year course in a cohesive manner without introducing so many new concepts as to be overwhelming.

This is a proof-of-concept design, not all elements are intended to be fully complete; there simply was not enough time available in the three months allocated to create a full module with the high level of quality expected from a third year BSc degree module. Therefore a proof-of-concept for one section of the module (assuming a module is made of two distinct sections each covering related topics) was to be created that can give a view of what could be possible with future work.

There were two possible outcomes of this project; either a section of the module with all the deliverables being draft examples of the resources

required for teaching and examining that section of the module, or a proof-of-concept of this section of the module could be built but with more time dedicated to each deliverable, such that each deliverable approached the quality required of a university module.

The initial planning for the project did not correctly estimate the work involved in creating the deliverables, whilst also managing to get the accuracy of the content to an acceptable standard. For this reason, while it was originally planned that a section of the module would be created, this was changed to a proof-of-concept a short time into the project. The key difference between a section of a module, and a proof-of-concept of the module is that a module section is meant to be a completed part of a larger module whereas a proof-of-concept is simply examples of different parts of a section. The intention with the module section was that it could be joined with another section to make a full module covering two subjects. In this project portions of the module section have been created, with those portions being indicative of the rest of the section that have not been fully created.

Chapter 1

Background

Cloud infrastructure management and cloud server configuration are both subjects that have received little to no attention from Cardiff Universities BSc Computer Science course. These disciplines are taking over the industry and replacing traditional on-premises infrastructure hardware and local manual software management. Essentially, the future is automated, virtualised, and in the cloud. The addition of a module or modules that cover this rapidly expanding field of computer science would help keep the Cardiff University Computer Science and Informatics course a the forefront of teaching and research in computer science.

There are presently few subjects currently taught on the course that would make either good companion modules, or could be merged with this projects proposed module. The computer science BSc teaches distributed systems as part of the ‘High Performance Computing’ and ‘Emerging Technologies’ modules, these would make good companion modules or could be augmented by this proposed module section. The National Software Academy (NSA) runs an Applied Software Engineering undergraduate degree course that already contains a DevOps module that focuses on Continous Integration and Continous Delivery (CI/CD) and cloud engineering; this proposed module section if augmented with a CI/CD or DevOps section could be comparable to the NSA’s module.

IaC is the name given to the managing of computing infrastructure (usually virtualised) through definition files. The cloud infrastructure is stored in text files that define the entire infrastructure in structured code-like blocks.

IaC is important to teach as it is becoming a foundational part of how modern technology and technology reliant organisations and companies

manage their various infrastructure suites. Modern software development demands an understanding of all parts of the software life cycle. Developers need to be familiar with the infrastructure that their code will run on, and operations teams need to be able to quickly change the infrastructure environments that they are managing in order to facilitate rapidly changing software requirements. IaC enables developers to have a hand in building the infrastructure that their code will be running upon, and also allow operations teams to quickly change running cloud environments to meet the needs of the business.

The main underlying reason for IaC being as prevalent as it is in the industry, is the widespread adoption of service based systems. These include Software as a Service (SaaS), Infrastructure as a Service (IaaS), and many other “as a service” systems. These systems are all run in the cloud and usually require subscription based payment models to access. As IaaS becomes even more prevalent, IaC becomes the obvious choice to manage the virtualised infrastructure that IaaS provides.

There are three main technologies that are referred to in this report. The first is Terraform, an Infrastructure as Code tool that creates and manages cloud and local infrastructure deployments in logical definition file stacks. From a programming perspective, Terraform code is simply object definitions that refer to one another. The Terraform program then constructs the infrastructure required, based upon these object blocks. The second technology is Ansible, unlike Terraform, Ansible is not truly an IaC tool as it is not used to create infrastructure from code. Ansible is a configuration management tool that is used to configure and install software on already running server instances. Ansible does this through so called “tasks”, a task at its most basic form is a function call to an Ansible method to make specific changes on the target machine(s). The third technology is Openstack; Openstack is a private cloud that, in this case, is hosted by Cardiff University. Openstack provides the virtualisation and the APIs that allow Terraform to build the infrastructure.

Chapter 2

Deliverables

2.1 Coursework

There are two tasks outlined on the pro forma (Appendix A) with each focusing on a specific aspect of IaC that is taught within the module section. The first task (Task 1) is designed to test the students knowledge and understanding of Terraform and infrastructure management. The task requires that the student builds an Openstack based cloud infrastructure that meets a specified set of requirements. The second task (Task 2) is designed to test the knowledge and understanding of Ansible and configuration management through IaC. The task requires the students to write a simple Ansible playbook that can configure the server instances created in Task 1 with webserver functionality.

The Terraform task (Task 1) is written in a verbose manner that explains to the student the theoretical layout of the new infrastructure that they are to build. This is done deliberately as it forces the student to piece together mentally the components that have been requested by the task, and then figure out how to write them as code blocks using Terraform.

The specifications are very strict about certain parts of the infrastructure, namely the number of security groups to use and the naming of the instances and SSH keys. The number of security groups is limited to one, this is to force the students to attach multiple rules to a single security group and therefore test their understanding (or at least give them an example to refer to in the future), of multiple rules in one security group. The naming of the instances and the SSH keys has no effect on the running of the Terraform code. However, forcing the students to use a specific naming scheme will make it much easier if an automated or semi-automated marking program

was to be used to assist the lecturer with marking the coursework.

The Ansible task (Task 2) is much smaller than Task 1, this is because the second task is much smaller in scope and complexity than the first task. The main reason for the relative simplicity of Task 2 is that this coursework is only designed to give a basic understanding of the subjects and technologies covered (IaC, IaaS, Terraform, Ansible, private cloud). Ansible can be used to make incredibly complex nested playbooks that can create intricate bespoke environments at scale. The advantages of requiring a bigger Ansible playbook would be minimal, as the students would only be adding more modules from the documentation. The basics however, how to make a playbook and provision a simple server, are incredibly useful and will allow the student to go on and learn the more complex parts of Ansible while already knowing the basics.

Task 2s requirements are very simple; the Apache 2 webserver must be installed onto the servers, the simple HTML file should be uploaded to the correct location, and all of the packages on the servers should be updated. The only more complex requirement in the task is for the playbook to request or receive the development server IP address when Ansible is run (as opposed to by storing the IP address in a file). This requirement is useful as it teaches the students that Ansible can be more than the configuration files that they have written. This is intended to demonstrate that whilst the configuration files are important, they are not the only way you can interact with Ansible.

The coursework pro forma Terraform task was originally written with multiple diagrams included, they were designed to help describe the tasks to the students, see figure 2.1. These images were simplistic as they needed to

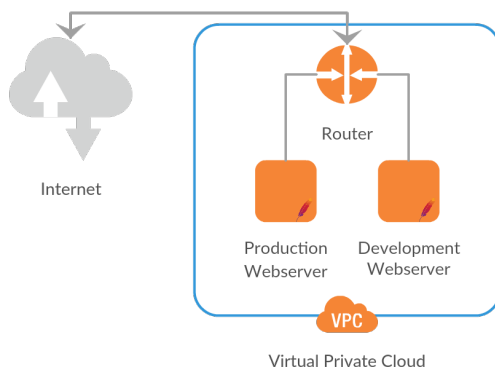


Figure 2.1: One of the diagrams originally intended for use in the pro forma

be understood by students who may have never had to try to build a cloud infrastructure from a diagram before. However their simplicity was their major problem, ideally the diagram would have every item of infrastructure that needed to be made with labels describing what they would all need to do. This level of detail would have made the diagrams incredibly messy to try to read. Whilst this may not have been to much of a problem for a more experienced cloud engineer, it would have made it very hard for the students to understand anything from the diagrams without it also being explained in text format elsewhere.

The text format to go along with the diagrams was written, however when it was finished the need for the diagrams fell away completely. The diagrams would take up valuable space in the pro forma and whilst they may be nice to look at and even helpful to a small minority of the students who had grasped the subject well, the decision was made to drop them from the pro forma entirely.

For both of the tasks outlined in the coursework pro forma, model answers were created to show what would be expected of the students. The full code repository for the answers is available at <https://github.com/domroutley/diss-coursework-answer>.

The marking criteria part of the pro forma lays out the task questions in a more rigid bullet point fashion. This is specifically designed to make sure that the students answers are easier to mark against a specific set of requirements. It also will make it simpler for students to understand exactly what they are being asked to create in the coursework.

The model answers given are just one way that the requirements in the pro forma can be achieved. Whilst the example lectures and the lab session both use the same sort of methods to achieve their results, a customary view of the Terraform Openstack official documentation ([9]) will reveal that there are often multiple valid methods of creating or linking certain resources together. The example answers assume that the students have followed the same sort of methods employed in the lab session and the example lectures, however other methods should also be viewed as valid.

The marking criteria does not limit the students to a specific method of answering the question, with one exception. It is requested that only one security group is used, this is so that the students are forced to use the networking services to create and manage the security groups. The non-networking method of managing security groups is messy and can be hard to read and understand, it therefore has been discouraged as it would be hard to mark correctly.

2.2 Lab session

The lab session is an example of one of the multiple lab sessions that would be run as part of the module section. As the majority of the coursework and the lecture completed to the highest standard is about Terraform, it was decided that the lab session to be created was to also focus on Terraform.

The lab session uses a GitHub repository to provide the students with some initial code that they are then tasked with editing in certain ways to teach them about the different Openstack modules and Terraform blocks. The repository that this code is stored in was originally built to provide screenshots of code for the lecture examples, this means that the examples given in the lectures and the lab sessions are practically identical. There are two repositories, a Terraform repository and an Ansible repository. The Terraform repository has instructions that can be used in a lab session, whereas the Ansible repository could later be used to create more lab sessions, but is currently just code that was used for screenshots.

The instructions for getting the Terraform code to run, are taken from the repository mentioned above, and some tasks to further the students understanding of the code were integrated into a lab session handout (Appendix B). The handout is split into three main sections; “initial system setup”, “running the module for the first time”, and “further tasks”. The “initial system setup” and “running the module for the first time” sections are slightly edited versions of the instructions included along with the code in the repository.

The initial system setup covers things that the student would need to do to get their machine ready for running the Terraform code for the first time. It contains three steps; downloading/cloning the code, generating an access token, and running `terraform init`. Generating the token is quite involved and will probably be the stumbling point for most students doing this lab session. For this reason a help file is included in the repository that goes through each step of generating a token in as much detail as can be reasonably included without overwhelming the student. The last step in the initial system setup is running `terraform init`, this initialises Terraform in the directory in which it is run.

The second section, “running the module for the first time”, walks the student through setting up the code to be able to run and produce a working infrastructure on Openstack. It is quite verbose, explaining each command to be run in sequence and also giving links to help pages for using the Cardiff University VPN, for those students who may be doing the lab session at

home or otherwise outside of the university network.

Finally, the further tasks section of the handout sets the student some simple tasks that will help with the coursework and their understanding of Terraform and Openstack. These tasks gradually ramp up in complexity, with the first task asking for just a couple of lines to be added to the outputs file, all the way to the third task that requests that the student adds a whole new server to the infrastructure. This third task lists all of the networking and access blocks that would be needed to successfully integrate this new server with the current infrastructure. This requires that the student focus more on the code they are writing rather than trying to figure out what they need to build, that challenge is left to the coursework.

2.3 Lectures

The main part of any module would revolve around the lectures and the lecture slides. This is where the vast majority of the content that a student is expected to know and understand by the end of the module is located. For this project, three sets of lecture slides were created to show an example of the sorts of content that would be delivered to the students taking the course.

There was limited time to spend creating deliverables for this project, so the lectures that were to be created had to be chosen carefully. For this reason the Ansible and Terraform lectures that were created are not completely indicative of the sort of content that would be in a full module. The idea is to have two lectures on each technology, the first being a more theory focused lecture talking about what the technology is for and why you would use it. With the second giving an example of the technology being used and showing sections of code to teach how to use the technology. With limited time to create lectures, and the lectures being much harder to create than was originally anticipated, the two lectures were merged into one. This has the effect of reducing the “unfinished-ness” feeling of the two lectures, while still delivering useable content.

2.3.1 Infrastructure as Code lecture

The Infrastructure as Code lecture is the shortest of the three lectures that was created for this project, but it was also the most focused in its goal. It has two objectives that the students should be able to achieve

after attending the lecture; knowing what IaC is, and knowing why we use IaC. The intention is that this would be the first “real” lecture (after the module description lecture) that the students would have. The lecture starts out by defining IaC and giving example of infrastructure and code, with a more precise definition of what exactly it means to manage Infrastructure as Code. There is then an overview of other important technologies and properties of IaC (Cloudformation, idempotence). The last five slides of the lecture are focused on, and concerned with, why IaC is important and why it is used in industry.

As has been mentioned above, there was limited time available which meant that only certain lectures were able to be made. The Infrastructure as Code lecture was deemed as integral to creating a useful proof-of-concept, it can serve as an overview of what is taught over the whole module whilst also showing a different style of lecture than the Ansible or Terraform lectures. Both of the Ansible and Terraform lectures are heavily focused on the “how”, with smaller “whats” preceding. The IaC lecture however, displays the “why” of Infrastructure as Code as a whole. The IaC lecture slides are included alongside this report, with an example slide being shown in figure 2.2.

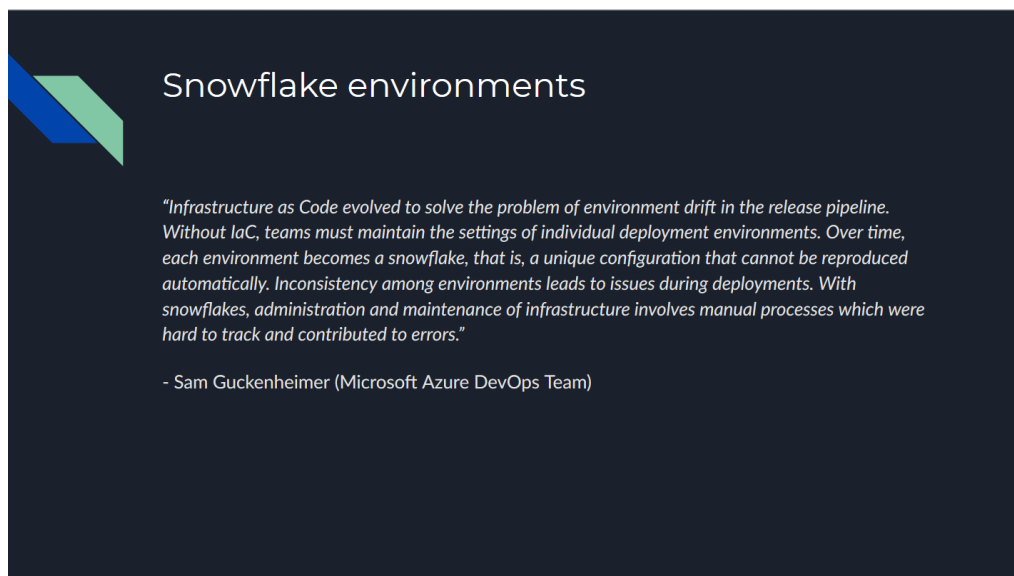


Figure 2.2: An example slide from the IaC lecture

2.3.2 Ansible lecture

The Ansible lecture, the first of the two technology explanation lectures, is the more undeveloped of the two. This lecture is a merging of two different lectures, the first part is focused very much on the “what” and the “why” of Ansible, with the second (much larger) part going over in depth an example of using Ansible. The lecture overall is mainly trying to teach a basic understanding of Ansible and how it can be used. The ideal plan would then be for a lab session covering Ansible to be held in the days after the lecture so that students can then apply what they have learned visually and audibly in a kinaesthetic manner. This will reinforce the learning and prepare them for the coursework and exam where they will apply this knowledge.

As the lecture was not fully completed due to time constraints, there are significant holes in the knowledge given by this lecture. The Ansible program itself is the main thing that has been omitted from this lecture. The lecture never talks about how to actually run an Ansible playbook, or what happens when a playbook is run. This has been omitted as while it is important, it is not required for the rest of the information (in the coursework and IaC lecture) to make sense.

The Ansible lecture slides are included alongside this report, with an example slide being shown in figure 2.3.

Host file titles allow you to group hosts and refer to multiple titles at once without having to write down the address multiple times.

This is done by using the :children suffix to the title, the titles of the “children” groups can then be listed

```
1 ---
2 [servers:children]
3 webServers
4 dbServers
5 cicdServers
6
7 [webServers:children]
8 europewebCluster
9 asiawebCluster
10 americawebCluster
11
12 [europewebCluster]
13 192.0.2.11
14 192.0.2.12
15
16 [asiawebCluster]
17 192.0.2.13
18 192.0.2.14
19
20 [americawebCluster]
21 192.0.2.15
22 192.0.2.16
23
24 [dbServers]
25 192.0.2.21
26
27 [cicdServers]
28 192.0.2.31
```

Figure 2.3: An example slide from the Ansible lecture

Both Ansible and Terraform are very important to the goal that this

proof-of-concept (or module section) is trying to achieve. However, while Terraform enables IaC through infrastructure management, Ansible complements this with highly specialised configuration management that needs to be understood by students studying IaC, this provides a contextual understanding of the place that IaC holds in the field of infrastructure management and configuration.

2.3.3 Terraform lecture

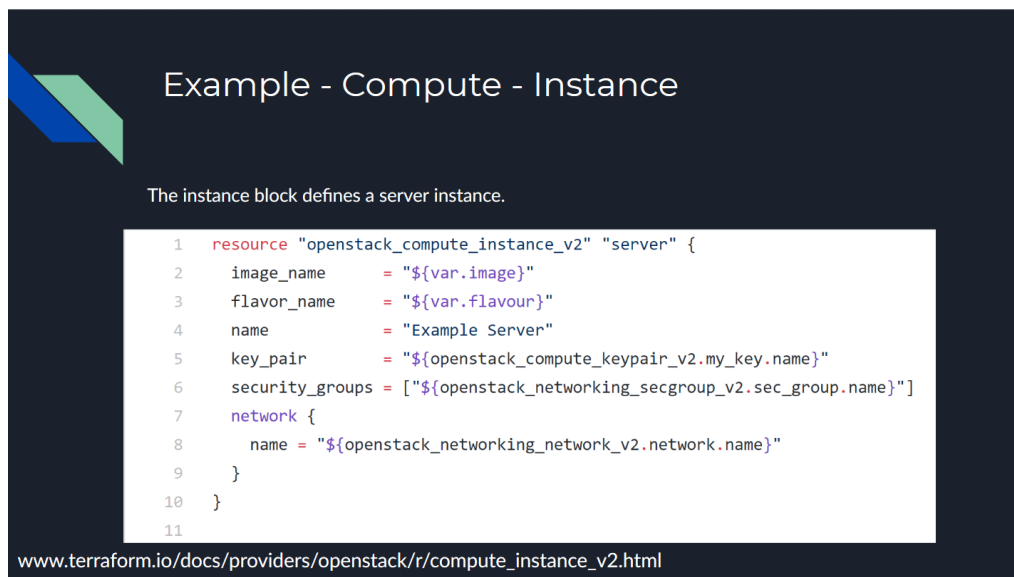
The Terraform lecture is the second and more fully developed technology explanation lecture. It has the same layout as the Ansible lecture, with the first, smaller, part containing the “what” is Terraform and the “why” use Terraform. The second part is then dedicated to going over, in depth, an example of how to use Terraform with code examples. As with the Ansible lecture, the Terraform lecture is trying to teach a basic understanding of its subject, and how it can be used in Infrastructure as Code. Unlike the Ansible lecture the lab session for Terraform has been completed, therefore we can clearly map the students learning progress from short term audio and visual learning, towards a more long term kinaesthetic learning method with the lab session.

This Terraform lecture is the most developed and largest of the three lectures that have been completed. Infrastructure management is the heart of IaC, and therefore Terraform is the primary focus of the module section. Again like the lectures discussed above, the Terraform lecture is not complete and is missing information that would be vital if the lecture was to be given as-is. However, unlike the Ansible lecture, the content currently in the Terraform lecture is at the standard that it is required for it to be presented to students as part of a module. For this reason, and to give an idea of how the lecture slides would be presented, a recording of the lecture being presented has been made and is available as a unlisted video on YouTube (www.youtube.com/watch?v=EM0gk-gn_Ys).

This recording has three main goals; to demonstrate how one of the lectures would be given, to show how a lecture slide can be broken down and explained, and to generate critical feedback on how the content of the lecture could be changed to make it better. A good example of breaking a slide down can be seen at 17:27 in the video, this slide is also shown in figure 2.4 below. The slide itself is simply an image of a definition block for a compute instance, with a single line of text stating as such. However each significant line of code, (in this case each key/value pair), is explained

in the video. This shows that there is more information included in the presented lectures than is displayed simply on the lecture slides.

The general (self delivered) criticism of the video is focused on one item, that the theoretical part of the lecture (the first part), was quite small and underdeveloped compared to the code example part (second part). Whilst it was known that this would probably be the case, due to the time constraint forcing that part to be reduced to be able to construct the code example part, it was only after the presentation that this became fully clear. But this feedback is good, it shows what needs to be worked upon to make the lecture better and ready for use in a module.



Example - Compute - Instance

The instance block defines a server instance.

```
1 resource "openstack_compute_instance_v2" "server" {
2   image_name      = "${var.image}"
3   flavor_name     = "${var.flavour}"
4   name            = "Example Server"
5   key_pair        = "${openstack_compute_keypair_v2.my_key.name}"
6   security_groups = ["${openstack_networking_secgroup_v2.sec_group.name}"]
7   network {
8     name = "${openstack_networking_network_v2.network.name}"
9   }
10 }
11
```

www.terraform.io/docs/providers/openstack/r/compute_instance_v2.html

Figure 2.4: An example slide from the Terraform lecture

The full Terraform lecture slides are included in a separate file alongside this report.

Chapter 3

Future work

At the moment the project contains most of what needs to be created for a proof-of-concept. This chapter, regarding future work, has been divided into two sections. The first focusing on completing the proof-of-concept. The second dealing with the work that would be required after the proof-of-concept is completed to turn it into a full module section or small independent module.

3.1 Proof-of-concept phase

The proof-of-concept phase was the work that was originally intended to be created by the end of the project, with the full module section then being knowingly relegated to this future work section. However, due to underestimation of the time that would be required to create the lectures, some parts of the original plan remain in a draft state, or otherwise not finished. This section will go over those deliverables that were not completed (or not even started) and discuss what would be needed to bring them up to the standard of completion required. The future work in this section will be discussed in descending order of importance, i.e if they were to be implemented then they should be implemented in the order that they are explained here.

The first and most important future task that should be prioritised over all of the others is the completion of the Ansible and Terraform lectures. These lectures are, as has been already mentioned in chapter 2 a merged amalgamation of two designed lectures, a theoretical overview/introduction to the technology lecture, and a code example/how to use the technology lecture. To finish these draft lectures they should be split back into their

composite parts as separate lectures and then fleshed out into full draft lectures. This would change the total number of deliverable lectures from three (IaC, Ansible, and Terraform) to five (IaC, Ansible theory, Ansible example, Terraform theory, Terraform example).

The main additions to be made would be made to the theory lectures as those parts are currently the most underdeveloped (even more so than originally thought when creating them, see the feedback on the Terraform lecture presentation on page 13). The obvious information that is missing and would need to be added to the lectures as a minimum, is the running of the programs (Ansible and Terraform), as at the moment there is no information about how to run the programs.

The Ansible lecture would also need to have the following additions:

- More information around the different supported Ansible products that are available
- An explanation of how Ansible actually works
- An example of the other Ansible files that are not currently covered in the lecture
- An overview of the best practice for using large Ansible projects with multiple playbooks with far more complex tasks

The exam deliverable was dropped very soon into the project, as it had been essentially designated as the “slip” deliverable. If there was not enough time to complete the project, then the exam was intended to be dropped first. This was for two reasons. Firstly, the exam questions and their model answers would not take too much extra time to create after the lectures and coursework had been created, as they would borrow heavily from those deliverables. The exam questions and the model answers would not really be able to feedback into the main knowledge base of the module, i.e the exam was not a generator of information, only a signifier or demonstrator of that knowledge.

The exam questions and the model answers would be focusing on the theoretical side of the module, in comparison to the coursework that would be focusing on the practical side of the module. An example of an exam question could be;

- Terraform and Ansible are both tools used as part of Infrastructure as Code, describe how you would use them together to create and manage a webserver on a cloud platform.

The student would then have to display knowledge of the differences between Terraform, an infrastructure management tool, and Ansible a configuration management tool. This question is simply an example of the sorts of questions that you would have in an exam to try to get the students to display the theoretical knowledge that they have learnt from the module. There would need to be a wide pool of questions to cover all of the subjects taught in the module, with the students being able to pick which questions they wanted to answer.

The last improvement that would need to be made to the project deliverables to finish the proof-of-concept phase is to increase the scope of the Ansible coursework task. Whilst it has been explained in chapter 2 that the Ansible task is deliberately uncomplex and smaller than the Terraform task. Using careful wording and changes made to the Ansible lectures, it should be possible to increase the coverage of Ansibles more complex features to a degree of understanding that can be included in the coursework. This would be useful for the exam questions also, as there would be a much larger amount of content that they are able to draw possible exam questions from and therefore increase the pool of questions that are able to be used. Furthermore, Ansible is currently a much smaller part of the module than Terraform. If the coursework (and by extension the lectures) for Ansible were increased in scope, both of the equally important technologies would be given a more equal representation in the module.

3.2 Creation phase of full module section

The first addition that would be made is an Ansible lab session. This lab session would have the same structure as the Terraform lab session that has already been created (Appendix B), but would be based upon the Ansible example code available on Github. This lab session would also have the same objectives as the Terraform lab session and would aim to be completable in one lab session. However it would probably be prudent to have both lab sessions run over two lab session time slots each so as to give the students ample help time to complete the tasks.

The other additions to the module would be the creation and organisation of the rest of the lectures needed to teach all of the information required for

the module. Assuming that the three lectures that are currently available have been scaled up to the five suggested in section 3.1, there will only be two lectures left to be created. Overall this will create a total of seven lectures, or roughly half the number of lectures for an autumn semester ten credit module.

The first new lecture to be created would be the module description lecture, this would not want to be that large or long, as it would probably have to share a lecture slot with the description lecture for the other section of the module. This lecture would include:

- The list of technologies and tools that would be covered in the module, making sure to note that these are not necessarily the only or the best tools, but they are the ones that give the widest base to learn from
- A reading list of potential books and/or websites containing extra information about IaC
- A timeline and short explanation of each of the lectures
- Information about the lab sessions
- The dates of coursework hand out and hand in
- Explanation of why this lecture is important and why the students should want to learn about IaC, configuration management, IaaS etc

The first of the “content” lectures would be the IaC lecture. Currently this lecture is in a draft state with much of the information missing that would need to be added to bring it to the required level of quality. Currently this lecture talks about what IaC is and why it is important, however it would also need to cover:

- Infrastructure as a Service
- Software as a Service
- Amazon Web Services
- Microsoft Azure
- Openstack
- Ansible

- Terraform

The second “content” lecture would be focused on Openstack. This lecture would aim to explain what Openstack is and give an example of what it can be used for. This example would be run using the web interface of Openstack, the idea behind the example given would be to create the exact same infrastructure that is created in the later Terraform example lecture to show the difference between creating a server manually and creating a server using an IaC tool like Terraform.

The fourth and fifth lectures would be the Terraform background and code example lectures (*see* 3.1, 2.3.3). They may need to be tweaked at this stage to make sure that they flow correctly from the previous Openstack lecture and do not create much of a split in the lecture style. The penultimate and final lectures will then be the Ansible background and code example lectures (*see* 3.1, 2.3.2). At this point the module *should* be complete and ready for delivering to a student class.

Chapter 4

Summary

Overall, this report was intended to facilitate the creation of a proof-of-concept university teaching module concentrated on cloud engineering and cloud operations. It was focused on creating prototypes of resources that could be used in a section of a module, centred around the cloud and web based industry. This chapter will summarise what was done to achieve the target of creating that proof-of-concept.

The first deliverable created as a prototype for this proof-of-concept module was a coursework pro forma detailing an example coursework that the lectures were based off of. The second deliverable was then the model answers created for the coursework, showing what is expected of the students when they are given the coursework. The next deliverable was the three lectures; one was a theoretical IaC lecture, and two were amalgamations of lectures designed for Ansible and Terraform. These two lectures merge the theoretical and practical parts of those technologies into a single lecture for each. While creating the lectures some code was written to provide examples, one of these code examples was then used to create a lab session for Terraform as the last deliverable. These deliverables are intended to be prototype-examples of what could be used if a full module, or module section, is created that is based upon the cloud engineering and operations fields that this report proposes.

This report also presents a plan for transforming the partially incomplete proof-of-concept into a complete proof-of-concept, and then into a fully complete module section that is ready to be delivered to students.

This project has succeeded in its primary goal of creating a proof-of-concept for a university teaching module based around cloud engineering and cloud operations. Not all deliverables that were originally planned were in fact

finished in time, mainly due to an underestimation of how time consuming certain tasks, (namely creating quality lecture slides) would be. The most important deliverables were prioritised and were completed to a high standard, enabling the proof-of-concept to be workable even if some of the less important deliverables were not completed.

It should now be possible for the deliverables to be used to create a full proof-of-concept module section, and then augmented with new content such that an entire module section can be completed. This module section could then be used along with a second related module section, to create a full ten credit module; or it could be delivered on its own as a smaller five credit independent module.

Chapter 5

Reflection

During this project; I have increased my technical skills regarding Openstack, and learnt about the difficulties involved in creating quality academic content.

Before this project, I had some limited knowledge of Openstack but I had never interacted with it. During this project I have gained a basic understanding of Openstack's web portal, mainly though interacting with it to retrieve information about the infrastructure stacks I was creating using Terraform. Most of the interaction I had with the creation and management of infrastructure on Openstack did not come from the web portal. Most of my understanding of Openstack came from using Terraform to interact with and control Openstack programmatically. My knowledge of Openstack is therefore centred around what Terraform can do with Openstack and not what Openstack is necessarily capable of. I also had to learn a bit about the Openstack API to enable me to use the API to retrieve an access token. My understanding of the API was then increased as I wrote a 'how to' guide for retrieving an access token.

When I started this project, I initially thought that the lectures would be relatively simple in comparison to writing the coursework and the subsequent coursework answers. This was not the case; I thought that creating the lectures would just be a case of using what I had done in the coursework and laying it out in slide format. I could then simply explain what was happening in each section of code. I am a very kinaesthetic learner, therefore I struggle to gain knowledge through the audio and visual methods provided by lectures, and instead learn while interacting with the subject. My learning style gave me a subconscious bias in favour of the coursework and practical learning, I was excited and enthusiastic to create the coursework and lab session deliverables, and thought that I could use them to create the lectures as opposed to creating the lectures first and building the coursework

from the content outlined in the lectures.

Looking back on this project with hindsight, I now realise that the order in which I planned to do the deliverables was not necessarily the best or the most efficient way to proceed. I think that if I had collected all of the relevant information together before writing the lectures, rather than gathering together extra theoretical information to augment the practical information that stemmed from the coursework, it would have made the organising and writing of the lectures much more time efficient. I would then have been able to build a series of draft lectures one after the other, that would have had all of the information and would be of gradually increasing quality, until they were completed.

What I believe I should have done was create a large content dump of the information that was going to be in the lecture, and then, when the content is all there, created the actual slides. Instead I created the lecture slide by slide, doing the research for each slide as I came to it. This was a highly time inefficient method of creating the lectures and resulted in the lectures taking longer than I had initially thought to create.

Appendix A

Coursework Pro forma

Outline

A small online news company, Generic Reporting, is using OpenStack to house their current cloud infrastructure. Their webserver is a basic Ubuntu 18.4 install, with Apache2 manually installed and configured onto it. Generic Reporting's new CTO thinks that their current cloud infrastructure is in need of an update. She wants a production server with an identical development server that can be modified/rebuilt at a moments notice.

Hand-in instructions

All submissions must be made via Learning Central. Upload the following files in a **single zip file named "CMXXXX_[student_number].zip"**. Make sure that the files are organised in the folder structure specified.

Hand in files

Description	Type	Name
Cover sheet	PDF (.pdf) file	[student_number].pdf
Terraform section	Terraform (.tf) files	provider.tf compute.tf networking_secgroup.tf networking.tf output.tf vars.tf
	SSH key file (no extension)	dragon pheonix
Ansible section	Configuration (.cfg) file	ansible.cfg
	Plain file (no extension)	hosts
	HTML (.html) file	file.html
	YAML (.yaml) file	main.yml

Folder structure

```

zip file
├── coversheet
├── terraform/
│   └── ...all 8 Terraform section files...
├── ansible/
│   └── ...all 4 Ansible section files...

```

Tasks

1. Using Terraform, construct the proposed cloud infrastructure as code from the description below.

The new infrastructure must contain two server instances that are identical in all respects other than names and SSH keys. These two instances should be both connected to a network along with a router that is also connected to the internet. There should only be one subnet with all three of these devices on it. The subnet DNS nameservers should be set to the University DNS servers. Using exactly one security group allow SSH, HTTP, and HTTPS connections to the subnet. All connections can be allowed from anywhere.

The server instances should be named “Production” and “Development”. “Development” should be allocated a dynamically associated IP address pulled from the University private floating IP pool. “Production” should have a statically associated IP address that is pre-reserved on OpenStack and will therefore result in the instance having the same IP address each time you create it with Terraform. This IP should **not** be written in any of the Terraform source files but should be stored in the “vars.tf” file.

There should be two different SSH keypairs for accessing the instances. One for “Production” and one for “Development”, the public keys for these keypairs should be stored in the “vars.tf” file. The “Production” keypair should be called “pheonix” and the “Development” keypair should be called “dragon”, both the private keys also need to be submitted.

There should be six outputs from the Terraform deployment, the IPs of each instance, the security groups of each instance, and the SSH key name of each instance.

Your Terraform code should be split into multiple files, each file should contain the following:

File name	What it contains
compute.tf	All blocks with resource type starting “openstack_compute”
networking_secgroup.tf	Resource types starting “openstack_networking_secgroup”
networking.tf	All other “openstack_networking” blocks
provider.tf	The provider block
vars.tf	All variable blocks
output.tf	All output blocks

2. Write an Ansible playbook that can provision the webserver from task 1 to the following specifications:

- Apache2 installed and running
- A file named file.html has been uploaded to the server in such a way that going to the address [IP]/file.html will return that file. (The content of the file is unimportant, just some text to prove that it works is sufficient)
- All packages in the system are up to date
- The dynamically assigned IP for the development server from task 1 is entered **at runtime** (entered when Ansible is run, not hard-coded in a file)

Marking criteria

Task 1

- Production server instance is associated with a static IP that is declared in the vars.tf file
- Development server instance is associated with a dynamic IP
- All servers should be on the same subnet and network
- The two webserver should not be accessible by anything other than HTTP, HTTPS and SSH
- Only one security group has been used
- Two SSH keys used. One for each webserver.
 - Production key named pheonix
 - Development key named dragon
 - Public keys stored in vars.tf file
 - Private keys authenticate correctly and allow access to server setup with public key
- The Terraform module should have six outputs.
 - The IPs of each instance
 - The security groups of each instance
 - The name of the SSH key used by each instance
- Terraform code split into multiple files as specified in the hand-in instructions.
- Effort has been made to variablise values that are not static

Task 2

- Only one IP is defined in the hosts file (production IP)
- A working method to input a IP to Ansible at runtime is used (command line args, prompt, ect)
- There are at least three tasks in the Ansible playbook that do the following:
 1. Update the system packages - Only this task is allowed to be accomplished with the raw module
 2. Install Apache2
 3. Upload a html file from the local machine to the servers and puts it into the /html/ directory of the Apache webserver
- All tasks successfully run on both server instances
- The uploaded file "file.html" is accessible on both instances from the "web". (Cardiff University internal network)

Appendix B

Lab session handout

Terraform initial lab session

Outline

This lab session will walk you through the steps to create a basic working Terraform managed cloud infrastructure as code. There will then be a series of tasks that will help you learn things that will be useful for your coursework.

If you are doing this lab session from home, then you will need to install Terraform from this link

<https://learn.hashicorp.com/terraform/getting-started/install.html>

Start here

Initial system setup

1. Download/clone this repository into a new directory
`https://github.com/domroutley/openstack-terraform-example`
2. Generate a token to access OpenStack and export it to your environment (for detailed instructions on how to do this open the `TOKENGENERATIONHELP.md` file from the repository you have just downloaded)
3. Run `terraform init` in the repository directory

Running the module for the first time

1. Assuming everything went ok with the setup, create an `ssh` key and copy the Public key into the `vars.tf` file as the default value of the variable `access_key`
2. You should now be all set. Run `terraform plan` to see the changes to be made to the cloud. `terraform plan` will also validate that your module is formed correctly and has no syntax errors.
3. If you are satisfied with what you see, run `terraform apply` and type and approve the changes.
4. You should be able to see the IP address of the instance as the output printed at the end of Terraforms run.
 - To bring this up again you can run `terraform output`
5. You can now SSH into the server with the following command
`ssh -i LOCATION/OF/SSH/KEY ubuntu@INSTANCE_IP`

- It may take a couple of minutes for the server to be able to accept SSH requests so be patient
- If you are not on the university network, you will need to connect via the VPN before you can SSH into the instance.
 - Using OpenVPN to connect to Cardiff Uni VPN Windows
<https://docs.cs.cf.ac.uk/notes/openvpn-windows>
 - Using OpenVPN to connect to Cardiff Uni VPN MacOS
<https://docs.cs.cf.ac.uk/notes/openvpn-macos/>
 - For Linux, get the ovpn file from one of the above links, then add a new connection in your network manager from the ovpn template file.

Further tasks

These further tasks will help prepare you for your coursework.

Don't forget to **terraform plan** after each task to test your syntax and structure, and then **terraform apply** to apply those changes to the cloud.

1. Add some more outputs to the module that output the DNS IP addresses
2. Add a new security group rule in the `networking_secgroup.tf` that allows HTTP access from all IP addresses, attach this rule to the already created security group and change the security group name to be more appropriate
3. Create a new server for FTP
 - Create the new compute resource resource
 - Create a new security group and security group rule that allow incoming connections via FTP (port 21)
 - Add the server to the already created subnet
 - Attach the SSH and HTTP security group to the new server
 - Attach the already created key pair to the new server
 - Create a new floating IP resource and attach it to the new server
 - Add outputs for the new IP, and outputs detailing what servers have what security groups attached

After you are done, run **terraform destroy** to remove your infrastructure stack from the cloud.

Glossary

Amazon Web Services

Amazon Web Services (AWS) is Amazon’s public cloud platform. 17,
see public cloud

Ansible

Ansible is an open-source software provisioning, configuration management, and application deployment tool. [3] It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates. [2] 4–6, 8–12, 14–19

Cloudformation

Cloudformation is an Amazon Web Services (AWS) product that takes a JSON-like “stack” of resources and creates the corresponding infrastructure in AWS public cloud environment. 10

Continuous Integration and Continuous Delivery

Continuous integration is the practice of routinely integrating code changes into the main branch of a repository, and testing the changes, as early and often as possible. Ideally, developers will integrate their code daily, if not multiple times a day. [4] Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way. [13] 3

idempotence

Idempotence is the property of a function/computer operation that can be applied multiple times without the result changing. e.g cancelling an order in an online shopping cart is an idempotent operation as no matter how many time the operation is run the result will always be that the order is cancelled. But placing an item in your shopping cart

is not idempotent as running the operation multiple times will result in multiple items placed in the cart, therefore being different each time. [5] 10

Infrastructure as a Service

Infrastructure as a Service (IaaS) is an instant computing infrastructure, provisioned and managed over the internet. [14] Infrastructure as a Service (IaaS) are online services that provide high-level APIs used to dereference various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning, scaling, security, backup etc. [6] They differ from traditional infrastructure as instead of the customer making a once off purchase of the hardware infrastructure and then possibly paying for further support, the customer only pays for the infrastructure that they are using while they are using it, with access to the (potentially virtualised) infrastructure being made via an API or web interface. 4, 17, 33, *see* Openstack & Software as a Service

Infrastructure as Code

Infrastructure as Code (IaC), is a type of IT setup wherein developers or operations teams automatically manage and provision the technology stack for an application through software, rather than using a manual process to configure discrete hardware devices and operating systems. [15] i, 4, 9, 10, 12

Microsoft Azure

Microsoft Azure is Microsoft's public cloud platform. 17, *see* public cloud

National Software Academy

The National Software Academy (NSA) is a partnership between Cardiff University, Welsh Government and industry leaders. [1] "The NSA at Cardiff University focuses on applied software engineering and learning technical skills like cloud engineering and managing virtualised server infrastructure through infrastructure as code." James Osborne [12] 3

Openstack

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter,

all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. [16] Openstack also has an API that can be used to control Openstack resources through an IaC tool such as Terraform. Cardiff University maintains an Infrastructure as a Service private cloud that is built in Openstack and is available for Computer Science students to use. 4, 5, 7–9, 17, 18, 21, *see* Infrastructure as a Service & private cloud

playbook

Ansible playbooks are expressed in YAML format and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process. Each playbook is composed of one or more ‘plays’ in a list. The goal of a play is to map a group of hosts to some well defined roles, represented by things Ansible calls tasks. [7] 5, 6, 11, 15, *see* Ansible

private cloud

A private cloud is a particular model of cloud computing that involves a cloud based environment in which only the specified client can operate. [17] Private clouds are often self-hosted by the client as opposed to hosted by the cloud provider as with public clouds. 4, 6, 33, *see* Openstack

pro forma

Pro forma/pro-forma/proforma (Latin for ”as a matter of form” or ”for the sake of form”) is in this case used to describe a document that is provided to describe the requirements for a section of work. It is the formal specifications of the work to be done. 5–7, 19

public cloud

The public cloud is defined as computing services offered by third-party providers over the public Internet, making them available to anyone who wants to use or purchase them. They may be free of charge or sold on demand, allowing customers to only pay per usage for the CPU cycles, storage or bandwidth they consume. [10] 33

Software as a Service

Software as a service (SaaS) allows users to connect to and use cloud-based apps over the Internet. Common examples are email, calendar, and

office tools (such as Microsoft Office 365). [18] Another good example of SaaS is G-Suite. Some definitions of SaaS include IaaS as part of SaaS, while others treat them as different but connected. 4, 17, *see* Infrastructure as a Service

Terraform

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. [8] Terraform is an open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp Configuration Language (HCL), or optionally JSON. [11] 4–12, 14–16, 18, 19, 21, 33, *see* Infrastructure as Code

Acronyms

AWS Amazon Web Services *Glossary:* Amazon Web Services

CI/CD Continuous Integration and Continuous Delivery 3, *Glossary:* Continuous Integration and Continuous Delivery

IaaS Infrastructure as a Service 4, 6, 17, 34, *Glossary:* Infrastructure as a Service

IaC Infrastructure as Code i, 1, 3–6, 10–12, 17–19, 33, *Glossary:* Infrastructure as Code

NSA National Software Academy 3, *Glossary:* National Software Academy

SaaS Software as a Service 4, *Glossary:* Software as a Service

Bibliography

- [1] *About us - National Software Academy - Cardiff University*
URL: www.cardiff.ac.uk/software-academy/about-us (visited on 05/01/2019).
- [2] *Ansible Documentation - Ansible Documentation*
URL: <https://docs.ansible.com/ansible/latest/index.html> (visited on 04/27/2019).
- [3] *Ansible (Software)*
URL: [https://en.wikipedia.org/wiki/Ansible_\(software\)](https://en.wikipedia.org/wiki/Ansible_(software)) (visited on 04/27/2019).
- [4] *Continuous integration, explained — Atlassian*
URL: www.atlassian.com/continuous-delivery/continuous-integration (visited on 05/01/2019).
- [5] *Idempotence*
URL: <https://en.wikipedia.org/wiki/Idempotence> (visited on 04/29/2019).
- [6] *infrastructure as a service*
URL: https://en.wikipedia.org/wiki/Infrastructure_as_a_service (visited on 04/27/2019).
- [7] *Intro to Playbooks - Ansible Documentation*
URL: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html (visited on 04/29/2019).
- [8] *Introduction - Terraform by HashiCorp*
URL: www.terraform.io/intro/index.html (visited on 04/27/2019).
- [9] *Provider: OpenStack - Terraform by HashiCorp*
URL: www.terraform.io/docs/providers/openstack/ (visited on 04/28/2019).
- [10] *Public Cloud - Definition*

- URL: <https://azure.microsoft.com/en-gb/overview/what-is-a-public-cloud/> (visited on 04/27/2019).
- [11] *Terraform (Software)*
URL: [https://en.wikipedia.org/wiki/Terraform_\(software\)](https://en.wikipedia.org/wiki/Terraform_(software))
(visited on 04/27/2019).
- [12] J. Osborne (Cardiff University)
Personal email conversation
May 1, 2019.
- [13] *What is Continuous Delivery - Continuous Delivery*
URL: www.continuousdelivery.com (visited on 05/01/2019).
- [14] *What is Iaas? Infrastructure as a Service*
URL: <https://azure.microsoft.com/en-gb/overview/what-is-iaas/> (visited on 04/27/2019).
- [15] *What is infrastructure as code? - Definition from WhatIs.com*
URL: <https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC> (visited on 04/27/2019).
- [16] *What is Openstack?*
URL: www.openstack.org/software (visited on 04/27/2019).
- [17] *What is Private Cloud*
Page now redirects to gtt.net, however searching for the quote will show it under the original URL
URL: www.interoute.com/what-private-cloud (visited on 04/27/2019).
- [18] *What is SaaS? Software as a Service — Microsoft Azure*
URL: <https://azure.microsoft.com/en-gb/overview/what-is-saas/> (visited on 05/02/2019).