A Web Driven SDN Orchestrator For The Provisioning of ACI Fabric and Lab Infrastructure

Matthew Gaynor

922830

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Portsmouth.

School of Computing
Engineering Project

2023

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Date: 2023

Abstract

This project will aim to provide an automation platform to CX Labs UK within Cisco Systems that will streamline DMZ lab operations by providing a web interface that will allow infrastructure to be managed from the perspective of the available rackspace. Cisco Application Centric Infrastructure shall be used to provide network connectivity and VMware ESXi and vCenter shall be used to provide compute resources. A testbed will be constructed to emulate a scaled-down version of lab infrastructure that will be used to develop and test the automation platform against Cisco ACI and vCenter. The platform will aim to obfuscate as much network configuration as possible, resulting in a drastic reduction in time spent provisioning lab rackspace when a project enters or terminates. This will result in a streamlined network configuration that represents the current state of usage, as the configuration will be managed entirely by the automation platform and derived from the user's inputs into the web interface.

Word Count: 11111

Acknowledgements

I would like to thank the CX Labs team, specifically my good friends Dave Smith and Kev Fountain for their continued support throughout this project, and for allowing me to use spare equipment to construct a testbed. Without them, this project would not have been possible.

Consent to Share

I consent for this project to be archived by the University Library and potentially used as an example project for future students.

Contents

Abbreviations

AAEP	Attachable Access Entity Profile
ACI	Application Centric Infrastructure
ACL	Access Control List
API	Application Programming Interface
APIC	Application Policy Infrastructure Controller
AS	Autonomous System
BD	Bridge Domain
CLI	Command Line Interface
CX	Customer Experience
DDI	DNS, DHCP and IP Address Management (IPAM)
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarised Zone
DNS	Domain Name System
DPG	Distrubited Port Group
DVS	Distributed Virtual Switch
ECMP	Equal Cost Multi-Path
EPG	Endpoint Group
ERD	Entity Relationship Diagram
FEX	Fabric Extender
HTTP	HyperText Transfer Protocol
LACP	Link Aggregation Control Protocol
LDAP	Lightweight Directory Access Protocol
MP-BGP	Multi-Protocol Border Gateway Protocol
MVC	Model View Controller
NAT	Network Address Translation
NIC	Network Interface Card
NOS	Network Operating System
NTP	Network Time Protocol
OOB	Out of Band
ORM	Object Relational Mapping

OSPF	Open Shortest Path First
RADIUS	Remote Authentication Dial-In User Service
REST	Representational State Transfer
SDK	Software Development Kit
SDN	Software Defined Networking
SPA	Singple Page Application
SSH	Secure Shell
STP	Spanning Tree Protocol
SVI	Switch Virtual Interface
SVS	Solution Validation Services
ToR	Top of Rack
VM	Virtual Machine
vPC	Virtual Port Channel
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding
VXLAN	Virtual eXtensible Local Area Network
WAN	Wide Area Network

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Client

This project will be developed with the intention of it being utilised by a client, for the benefit of their operation. The client is CX Labs UK within Cisco Systems. CX Labs provides lab space for use by business units internal to the company. Most of the space is used for the testing of customer networks by SVS (Solution Validation Services). SVS provides bespoke testing services to customers wishing to use Cisco's expertise to test a range of scenarios, from regression and firmware testing to full upgrade and migration plans.

To match the customer's environment as close as possible, a scaled-down version of the customer's network is usually recreated in the lab space managed by CX Labs. CX Labs hold many devices that cover most of the Cisco portfolio, which allows for the recreation of most networks. Most of this lab space is hosted within the internal Cisco corporate network which requires any users to be employees of Cisco to access testbeds. More and more customers however are requesting remote access to their testbeds so they can perform their own testing. To facilitate this, a fully isolated DMZ environment is provided, which allows direct WAN connectivity to a testbed, allowing for a VPN tunnel to be established and hence remote access granted to a testbed from any location to any permitted person.

Current Infrastructure

The current infrastructure to support the testbeds consists of four Nexus 9K core switches, with FEXs to provide copper connectivity. Currently, the N9Ks do not have any form of vPC configured, and just use STP for redundancy. ToR connectivity is provided by Catalyst switches. Each project has a dedicated VLAN to isolate communications between different projects, and a virtual router and services stack is hosted on a vCenter environment to provide internet and VPN connectivity to the project. Each project also has several terminal servers attached to its VLAN so that the console ports of the testbed

devices can be easily accessed in case of a device problem.

Current Project Pipeline

When a new testbed build is requested, the project topology is reviewed, which will indicate how much rackspace will be required to accommodate the project. Suitable racks will then be chosen to house the project based on cooling, power and space requirements. A new VLAN for the project will then be manually created across all of the associated networking equipment, including the core N9Ks, and the ToR switches that live in the selected racks. Terminal servers will also have to be reconfigured so that they have the correct subnet configured and the correct sub-interface configured. The next step is to provision a virtual CSR1000v router hosted on vCenter that will provide NAT and internet access to the newly created VLAN. A services stack will then be deployed to take care of remote access VPN and other associated services. Both of these steps require the manual addition of a DPG to the DVS present in vCenter.

1.2 The Problem

Whilst the existing solution does function, several problems frequently arise which reduce the operational efficiency of the lab:

A large initial time investment is required when onboarding a new project. This is due to the manual configuration of the networking equipment, as well as the manual deployment of the virtual router and services stack. This time could be better spent on other tasks, such as preparing the physical rackspace for the racking and stacking of new equipment.

No configuration management system is in place, which results in configuration drift over time. When projects are removed, the configuration is sometimes inconsistently modified on devices. Both VLANs and DPGs become unsynchronised and often differ between the core switches themselves, resulting in confusion when modifying configuration. In the past, this has led to accidental removal and modification of configuration related to other projects, which impacts customer availability and takes time away from the team that could otherwise have been used to tackle other issues.

No centralised device management is utilised which results in device health and firmware being hard to update and monitor. This is because each device has to be manually updated, which is a time-consuming process. This also means that the devices are not all running the same version of firmware, which can cause issues when troubleshooting. A lack of centralised management makes device failure also hard to detect as there is no centralised monitoring solution in place, so reports from the testing team or customer are often the first indication that a problem has arisen.

1.3 Aims and Objectives

This project aims to provide a solution that will automate the configuration of the DMZ network infrastructure, as well as the configuration of the associated project infrastructure, such as the project router, services stack and terminal servers. This will be achieved by providing a web-based dashboard that allows the user to provision a new project, which will then automatically configure the required infrastructure. The solution will revolve around recreating the physical rackspace inside the web interface. The concept behind this is a one-to-one mapping between the real-world rackspace and the virtual rackspace. The virtual rack can then have the ToR and terminal server that resides in the real-world rack associated with it. A project can then be allocated virtual racks and have the associated infrastructure that is tied to a rack automatically provisioned. The solution will also provide a view of the current utilisation of the lab space by projects, as well as the current utilisation of the lab space as a whole.

The solution will use Cisco ACI to provide network connectivity. This removes the complexities that would otherwise be associated with provisioning many network devices via SSH or RESTCONF. ACI will also make the network highly scalable, with the ability to add additional leafs and FEXs to support any expansion of the rackspace. VMWare vCenter will be used to host virtual machines associated with project infrastructure, such as the project's virtual router and services stack.

This report will detail the design and conception of a testbed that will be used to test the automation platform against ACI and vCenter. The testbed design can then be used to influence the design of a fully-fledged fabric and compute infrastructure to support testing in the future.

1.3.1 Deliverables

- A web-based dashboard that allows the user to manage and provision projects
- User guide for how to provision ACI, vCenter and other associated network infrastructure so that it will be compatible with the automation solution - Appendix D
- User guide for the dashboard - Appendix E
- This report, detailing the design and implementation of the solution

1.4 Limitations and Risks

As the testbed that will be used to test and develop the solution is in a remote datacenter, physical access to the testbed is limited. This may result in delays if a problem that requires physical remediation occurs. The testbed will be designed to be resilient through the use of redundant power supplies and links where possible. If a failure does occur,

then the project may run over time and key milestones may have to be adjusted accordingly. With the testbed being hosted remotely, there is also a possibility that access to the testbed may be revoked, which would result in the project being delayed until access is restored. External factors such as the availability of the testbed's internet uplink and the power supply may also have an impact on the delivery of the project within the required timescale.

Chapter 2

Literature Review

The aim of this literature review is to research and analyse existing solutions, documentation and research on the automation of networking infrastructure. To ensure this review is of maximum usefulness to the development of the solution, it will also involve analysing best practices and standards when developing software and automation solutions. This will allow for the optimisation of the planning and implementation stages of the project that will subsequently follow.

Sources utilised for this review will be relevant books, online websites, professional publications and Request for Comments.

The research performed was related to the following set of topics:

- What is software defined networking?
- What types of software defined networking exist?
- What automation is currently used in the networking world and why?

2.1 Definition of Software Defined Networks

Industry experts and academics define software defined networking similarly, that is, providing automation and intelligence to networks via the means of software and APIs. Kreutz et al. (2015) state that "SDN was originally coined to represent the ideas and work around OpenFlow at Stanford University".
Four pillars are often used to define the differences between conventional networking and OpenFlow. Kreutz et al. (2015) define these as the following:

1. The decoupling of the control and data planes.
2. Forwarding decisions are based on flows, which represent a set of packets with the same characteristics.

3. Decision-making logic is moved to a centralised controller which has visibility over the whole network.

4. Providing the ability to programmatically interact with the network through the use of APIs and SDKs.

Whilst these four pillars illustrate positive aspects of OpenFlow, it has many scalability disadvantages that have led it to become less favourable when deploying larger networks (Alsaeedi et al., 2019). As a result of OpenFlows shortcomings, SDN has been divided into two subcategories, imperative and declarative (CDW, 2015). Imperative SDN is where "A centralized controller (typically a clustered set of controllers) functions as the network's 'brain"' (CDW, 2015) and declarative SDN is where "the intelligence is distributed out to the network fabric. While policy is centralized, policy enforcement isn't" (CDW, 2015). Using this definition, OpenFlow can be placed into the imperative SDN category, as the controller is used to directly influence the packet forwarding process (Kreutz et al., 2015). With SDN now referring to different methods of making networks smart, a concrete definition has become harder to reach. SDN is referred to as "an innovative architecture that separates the control plane from the data plane to simplify and speed up the management of large networks" (Wazirali et al., 2021), however, other researchers take a more programmatic view of SDN. An alternative definition of SDN is "a new networking architecture that is designed to use standardized application programming interfaces (APIs) to quickly allow network programmers to define and reconfigure the way data or resources are handled within a network" (Kirkpatrick, 2013). Whilst these are two different perspectives, modern SDN is a combination of both, with both programmable APIs and a decoupled control and data plane both being features of SDN.

In summary, a SDN is a network architecture that separates the control plane from the data plane and provides automation and intelligence to networks through software and APIs, whilst providing a centralised point of administration to the network administrator.

2.2 Declarative vs Imperative SDN

As briefly touched upon in the previous section, SDN is broken up into two main types, imperative and declarative. The imperative and declarative definitions originally originate from software development. Imperative programming is the traditional programming method, where the programmer specifies all steps in order to achieve a desired outcome (Latif et al., 2020). Declarative programming, however, is where the overall goal is specified, instead of all of the intermediary steps that must be completed to achieve the goal (Latif et al., 2020). Translated into networking, imperative SDN perfectly explains what OpenFlow set out to do, and that is for the controller to make the decision and inform the end networking device exactly how to forward and handle the packet. Since the creation

of OpenFlow, many declarative solutions have been created to provide a more scalable solution. Declarative solutions such as OpFlex are designed for "transferring abstract policy from a modern network controller to a set of smart devices capable of rendering policy" (Bhardwaj, 2020). Bhardwaj goes on to state how "OpFlex is designed to work as part of a declarative control system such as Cisco ACI in which abstract policy can be shared on demand." Another declarative protocol is NETCONF, which allows for device configuration to be read and modified through the use of Remote Procedure Call (Latif et al., 2020).

2.3 Why Use Software Defined Networks

Since this project is centered around SDN and the automation of these networks, it is critical to ensure that the principles and their method of operations are understood, and the benefits of using it.

An official survey paper from the IEEE that analysed the state of SDN provides a good explanation as to why the need for network programmability arose in the first place. Conventionally, "Computer networks are typically built from a large number of network devices such as routers, switches and numerous types of middleboxes" (Nunes et al., 2014). Nunes et al. goes on to state that due to the large amount of manual configuration required to achieve the desired traffic flow, "network management and performance tuning is quite challenging and thus error-prone". Having a centralised controller allows for a single point of management (Seyedebrahimi et al., 2016), which makes the management of a large network much easier, as is found with SDN environments.

As referenced earlier, the ability to programmatically control and interact with a network is also a key benefit of SDN. APIs "quickly allow network programmers to define and reconfigure the way data or resources are handled within a network" (Kirkpatrick, 2013). A northbound API provides a "high-level API between the controller and the applications" (Zhou et al., 2014) which need to interact with the network.

2.4 Software Defined Networking Solutions

As expected, many manufacturers have released and developed solutions that use the principles of SDN to automate network operations using a variety of hardware. This section will explore the different types of SDN solutions that are available in the industry.

Cisco ACI

Cisco ACI is a proprietary solution from Cisco that uses the Nexus 9000 series of switches using a special firmware version. Whilst the design of the datacenter fabric remains essentially unchanged, with spine-and-leaf being the required design, where ACI does introduce change is with how configuration and policy are applied to the networking devices. Cisco ACI still utilises the leaf-spine architecture which has been proven to be

highly scalable and provide the data throughput that is required for modern datacenters (Alizadeh & Edsall, 2013). The whole fabric is Layer 3 so that ECMP can be utilised to share load across multiple links, however, overlay protocols such as VXLAN are utilised to allow any workload to exist at any point in the fabric (Duffy, 2014). ACI also features plug-and-play fabric discovery, where new switches are automatically discovered by the controller and can be onboarded with ease, making future network expansion very easy to achieve.

Juniper Apstra

"Juniper's Apstra solution provides a deployment method called connectivity templates, which allow administrators to create and reuse validated templates to set up multi-vendor networks. It supports multiple device operation systems, including Cisco NX-OS, Nvidia Cumulus and Juniper Junos OS" (Xu & Russello, 2022). The main advantage Apstra has over Cisco ACI is the fact that it supports multiple vendors. This prevents becoming locked in with a vendor's future and current hardware portfolio. Apstra is still in its infancy, as it was only released in December 2020 (Xu & Russello, 2022), which means that documentation and training material are still sparse, and it has not been proven in the field to be as reliable in a mission-critical environment as Cisco ACI.

VMWare NSX

VMWare NSX is a software-defined networking solution that is designed to be used in a virtualised environment. Whilst this provides many advantages for improving networking when using virtual machines and applications, NSX provides no management for physical networking, and is purely focused on providing automation and networking for applications. It mainly provides tools and telemetry for day-two operations and troubleshooting (Ijari, 2017). This means that whilst NSX is a good solution for virtualised environments, it is not suitable for physical networking deployments.

2.5 Software Defined Networking Alternatives

Whilst SDN is a great solution for automating networks, it is not the only solution. This section will explore the alternatives to SDN and their advantages and disadvantages.

Ansible

Ansible is an open-source piece of software from Red Hat that is described as a configuration management tool (Wågbrant S, 2022), where a form of state description can be written, and then verified through the use of Ansible (Borgenstrand, 2018). Whilst Ansible does not provide the same level of automation as SDN, it does provide a way to automate the configuration of network devices, and can be used to automate the deployment of new devices. The network configuration must be designed and built before Ansible is used to push configuration to the devices. Ansible does help solve issues of scale and

complexity, as it can be used to push configuration to multiple devices at once, and can be used to automate the deployment of new devices. Ansible can also be used to automate the deployment of SDN solutions, however, it does not provide a turn-key configuration solution and still requires the design of the network configuration to be specified (Collin, 2021).

Chef

Chef is another open-source configuration management tool that is similar to Ansible. Chef can easily handle up to 10000 nodes from a single chef server (Sabharwal & Wadhwa, 2014), however, it is designed to use an agent which must be installed upon the device to be automated which adds another layer of complexity. It also requires the design of configuration and is useful only for deployment and preventing configuration drift.

2.6 Developing Software To Interface With SDN

SDN already provides the facilities to programmatically interact with the network through the use of REST APIs. Most solutions provide a 'northbound' API which can be used by developers to interact with the policy defined in the SDN controller. Any changes made via this API will then be propagated down to the network devices, with no interaction with the actual devices themselves required. The northbound API allows developers to focus on controlling the SDN instead of worrying about sending actual commands down to the network devices (Vasconcelos et al., 2017). There is no standard to northbound APIs and they are heavily vendor specific (Pham & Hoang, 2016), this means that an application is only able to support one SDN platform, unless compatibility for multiple is explicitly developed and accounted for. This means that the correct platform should be chosen in advance of a solution being developed.

2.7 Disadvantages of SDN

Whilst the benefits of SDN have been discussed, it is not without its shortcomings. One of the big issues with SDN is choosing a solution, as vendor inter-operability is still a significant drawback of SDN (Sokappadu et al., 2019). Once a solution is selected, it is very difficult to migrate to another or add hardware from another vendor. Security is also hard to find as part of direct integration with most SDN solutions, which results in multiple control planes, one for data and one for security (Sokappadu et al., 2019). This results in multiple points of administration and a lack of consistency in configuration style and integration between the networking and security elements. SDN also relies on centralised controllers, whilst these controllers are not critical for traffic flow in declarative settings, they remain a centralised point of failure that could lead to downtime (Rana et al., 2019). This centralised nature also ties into making the controller a target for DDoS attacks, as taking out the controller would result in disruption to wider network operation (Yan et al.,

2016).

2.8 Existing Automation Solutions

Existing automation solutions do exist in the industry, one example of this is the Open Distributed Infrastructure Management platform. This aims to provide a unified management platform for a variety of hardware and software from a variety of vendors (Networking, 2021). The Resource Aggregator for ODIM (Project, 2023) provides the actual integration ability that allows for unified viewing of infrastructure and the ability to "manipulate groups of resources in a single action". Whilst this does not directly solve the problem the client faces, it provides an easier way to automate a collection of infrastructure. Terraform is also another solution that provides a way to define infrastructure as code (Terraform, 2023). This allows for the definition of infrastructure configuration in a declarative way, and then Terraform will handle the deployment of the infrastructure. Terraform also supports the automation of vCenter, which is the virtualisation platform that the client uses. This means that Terraform could be used to automate the deployment of virtual machines and the configuration of the ACI fabric, however, it would not as a whole provide a turnkey solution to the problem and would require configuration and planning to use. This is because Terraform maintains the state of the infrastructure based on the configuration files provided to it (Collin, 2021), and does not provide a user-friendly method of interacting with the infrastructure.

2.9 Conclusion

Software Defined Networking uses a modern approach to improve the efficiency and scalability of conventional networking. Through the use of centralised control and administration, network administrators no longer have to develop such advanced scripts or have as large a workforce to manage and maintain large networks, that can often have thousands of devices. Declarative SDN solutions seem to be the way forward, as they take all of the advantages of having centralised control and administration, but don't place as heavy a load on the centralised controller when it comes to influencing packet flow and routing decisions. The ability to easily develop software to control network connectivity, without any of the fuss of integrating and managing any devices that provide physical connectivity is a big bonus and will be imperative to the success of the project.

Whilst there are automation solutions that allow automated configuration and deployment, they all require an element of manual configuration and do not provide a user-friendly way of providing the required data that would be needed to deploy the correct configuration to the network. As the problem is highly specific to a testing environment, there is currently no alternative solution that provides the exact feature set that is desired.

Chapter 3

Methodology

In this section, the methodologies used to develop and test the software will be detailed. This will include the tools used, the development process and the testing process. As the primary artefact will be software-based, it makes sense to select an appropriate development cycle that will ensure that the artefact is delivered on time and to specification.

3.1 Development

Whilst there are many software development approaches, Kanban was chosen as the best methodology because only one person will be working on the development of the software, not a team of developers. Kanban is a simple and effective way to manage a single person's workload and is a good fit for this project. A solution such as Trello, (Atlassian, 2023), can be utilised as it is free for small teams and is quick and easy to use. The Trello board that will be used for development is shown in figure 3.1.

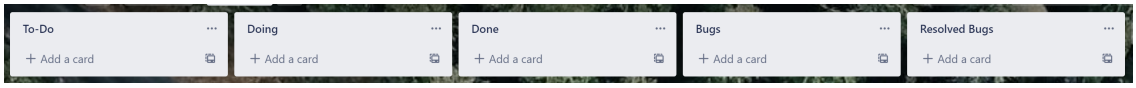

Figure 3.1: Trello Kanban board used to manage the development process

The project requirements will be broken down into smaller tasks, and each task will be added to the Trello board. The tasks will be added to the "To Do" column and then moved to the "Doing" column when the task is being worked on. Once the task is complete, it will be moved to the "Done" column. This will allow the project to be broken down into smaller tasks and will allow the project to be managed effectively. The tasks will be added to the board in the order that they are required to be completed, and will be worked on in that order. This will ensure that the project is completed in the correct order and will ensure that the project is completed in the required timescale. There are also 2 other lists, namely "Bugs" and "Resolved Bugs" which will allow for the ease of tracking bugs and ensuring that they are resolved during the development cycle.

3.2 Time Management

To ensure that the project keeps to time, a Gantt chart is to be used which will outline the key milestones of the project and when they should be met. To generate this chart, (teamgantt, 2023), was used as it has a free tier and meets all of the requirements for this project. Figure 3.2 shows the Gantt chart for the project.

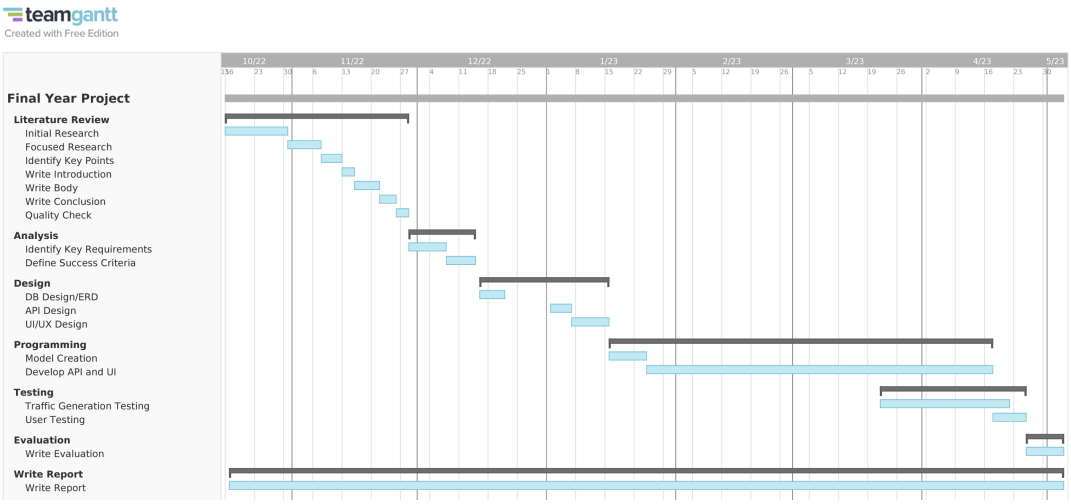

Figure 3.2: Gantt chart used to outline the key project milestones

As the project will follow the Kanban development methodology, some stages overlap and will occur simultaneously. This is because testing will be critical to influencing the development process and ensuring that features work correctly as they are developed. The tight timescale of this project also means that tight features will be tested as they are written to prevent issues later in the project.

3.3 Project Management

To ensure secure storage, accessibility and history of code as it is written, Git will be used to manage the project. Git is a version control system that allows for the tracking of changes to code and the ability to revert to previous versions of the code. GitHub will be used to host the code and will allow for the code to be accessed from anywhere; GitHub also has a free tier which will be used for this project. The GitHub repository for this project can be found at *https://github.com/mattg66/fyp*.

Chapter 4

Requirements

This chapter will provide detail on the requirements for the automation platform. The requirements will be split into 2 sections, namely the functional requirements and non-functional requirements. The functional requirements will outline the features that the platform must have and the non-functional requirements will outline the requirements that the platform must meet to be successful. To prioritise the requirements, the MoSCoW method will be used. This method will allow for the requirements to be prioritised and will ensure that the most important requirements are met first. The MoSCoW method is a prioritisation method that splits requirements into 4 categories, namely: Must Have, Should Have, Could Have and Won't Have.

4.1 Functional Requirements

Functional requirements are outlined by the IEEE as a "function that a system or system component must be able to perform". ("IEEE Standard Glossary of Software Engineering Terminology", 1990)

ID	Details	Priority
FR1	Visual representation of rack space	Must Have
FR2	Add and remove racks from the space	Must Have
FR3	Add and remove Terminal Servers from racks	Must Have
FR4	Add and remove Fabric Nodes from racks	Must Have
FR5	Add, remove and update projects	Must Have
FR6	Expand or contract a projects consumption of rack space	Must Have
FR7	Automate configuration of ACI fabric	Must Have
FR8	Deploy virtual router using vCenter API	Must Have
FR9	Deploy virtual services stack to provide remote access VPN	Could Have
FR10	Continuous monitoring of ACI and vCenter health	Won't Have
FR11	Terminal server automated management	Must Have
FR12	Login system to restrict access	Could Have

Table 4.1: Functional Requirements

FR1 - FR2 outlines the requirements to have the rack space visualised in the web application of the solution. The idea behind this is that the application will simplify the process of adding and removing racks which will allow for the rack space to be easily recreated, and will also allow for the rack space to be easily updated if the rack space changes. It will also help show the utilisation of the space, and allow for project planning to be carried out more easily.

FR3 - FR4 outlines the ability to associate ACI nodes and terminal servers to racks, this is required so that the automation backend can push the required config out when a rack is onboarded into a project. This also adds the ability to add and remove nodes and terminal servers if any physical changes occur in the rack space.

FR5 details the ability of the automation platform to store projects. This will provide the core functionality of the platform, where the current projects are stored and managed through the automation platform.

FR6 details the requirement to expand and contract a project's rack space utilisation, this will allow for the project to be scaled up or down as required which is a common occurrence.

FR7 outlines the core automation functionality of the platform. This is to automate the

deployment of connectivity to the ACI fabric based on the selected rack space and associated fabric nodes.

FR8 outlines the deployment of a virtual router to the vCenter automation platform. This will provide internet connectivity to the project network created by FR7.

FR9 provides the ability to automate the creation of a project services stack, this may include services such as VPN and NTP to name a few.

FR10 outlines the possibility of having continuous status monitoring of ACI and vCenter, however, due to the required time to implement this feature it has been marked as a Won't Have.

FR11 details the ability to also automate the terminal servers associated with racks which will ensure that terminal servers are connected to projects upon their onboarding.

FR12 details the possibility of implementing a login system, whilst this would be a useful feature and should be implemented at some point, the project will be hosted on a secure network that requires access to be granted, so the login system may be out of scope given the time restrictions.

4.2 Non-Functional Requirements

Non-functional requirements describe the non-behavioral characteristics of a system, capturing the properties and constraints under which a system must operate (Antón, 1997)

ID	Details	Priority
NFR1	Must be easy to use for staff with less technical knowledge	Must Have
NFR2	The system status should be easily visible to staff (e.g. errors, project status)	Must Have
NFR3	The system should be able to easily integrate with existing ACI fabric deployments	Could have

Table 4.2: Non-Functional Requirements

NFR1 outlines the requirement for the web application to be easy to use for less experienced team members. Through the use of abstraction, the networking and configuration can be hidden behind an easy to use web interface through the use of automation scripts that are run as a result of the user's actions.

NFR2 shows that the system must report the status to staff via the use of status indicators. This should show the progress of the automation scripts as they progress through automating and applying the configuration to various elements of the network.

NFR3 outlines for the platform to be able to integrate with existing ACI fabric deployments. This will allow for the platform to be used in a production environment without the need to reconfigure and rearchitect the fabric. Ideally, the platform should be deployed alongside a new fabric in a greenfield deployment.

Chapter 5

Design

As the project has utilised an iterative approach with the Kanban methodology, the design has changed and been refined throughout the project. The design of the software will be broken down into 3 sections, namely the design of the web application and the design of the ACI and vCenter configuration. The design of the testbed will also be detailed.

5.1 Web Application

5.1.1 Architecture

A client-server architecture will be utilised to provide the user-facing experience, and the automation and data handling logic hosted on the server. By using this model, many clients can request and interact with data that is hosted on one central server. The backend and frontend that make up the application will be separate from one another, and will therefore be developed independently, with the frontend interacting with the backend via a REST API. This allows for the front end to be a SPA, which facilitates a better user experience due to the lack of page refreshes upon every request.

The server will process all requests generated by the front end and also make requests to the various APIs that will be required to automate the network deployment. The database will also store all data required by the server to generate the appropriate ACI and network configuration that is required to automate the deployment of projects to the ACI fabric and associated network devices.

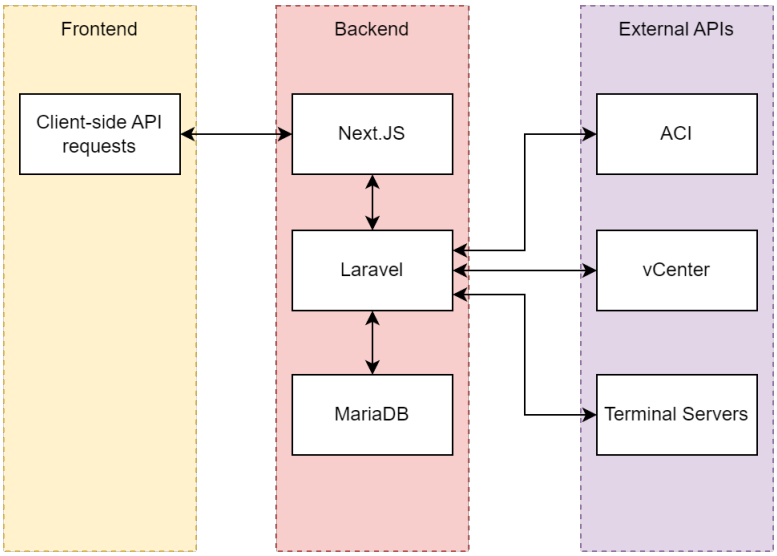

Figure 5.1: Web Architecture Design

5.1.2 Frontend

Next.js will be used to power the frontend of the application as it is an enhancement of React.js and provides server-side rendering and acceleration of pages. React uses a modular 'componentised' approach to building the frontend, which allows for the creation of reusable components that can be used throughout the application. This allows for the creation of a modular and scalable frontend that can be easily extended and maintained.

React.js also has an extensive library of open-source components and libraries that can be utilised to make developing the frontend easier and more feature complete. Due to the complexity of some required features, such as having a drag-and-drop interface for the recreation of the lab space, the use of a library such as React Flow will be required as the time required to develop such a feature would be out of the scope of this project.

Tailwind CSS and Flowbite will be used to style the web application. Tailwind CSS allows for the creation of custom components and styles that can be reused throughout the application. Flowbite is a UI toolkit that is built on top of Tailwind CSS and provides a set of pre-built components that can be used to accelerate the development process. Tailwind CSS also provides features such as native support for dark mode, which is a feature that is becoming more popular in modern applications.

5.1.3 Backend

The backend of the application will be written in PHP, using the Laravel PHP framework. Laravel is a popular PHP framework that will accelerate the development process, as it features an inbuilt ORM, API routing system and an authentication system that can be implemented. Laravel utilises SQL-based databases, and as such MariaDB will be

used as the database for the application. Laravel also features an in-built HTTP client which will be required to interact with the ACI and vCenter APIs which are all REST-based.

Laravel follows the MVC architecture, however as the frontend is a React SPA, Laravel will only be used to provide and consume the data via its REST API.

Laravel also features a queuing system which allows for jobs to be processes asynchronously without affecting the process responsible for handling API calls from the web interface.

5.1.4 Database Design

As the project will need to store data persistently, ensuring that the database has an appropriately designed schema will ensure that the data is stored in a way that is easily accessible and can be queried efficiently. The design of the database was tweaked and refined throughout the development process, however, the final ERD is shown below in Figure 5.2.

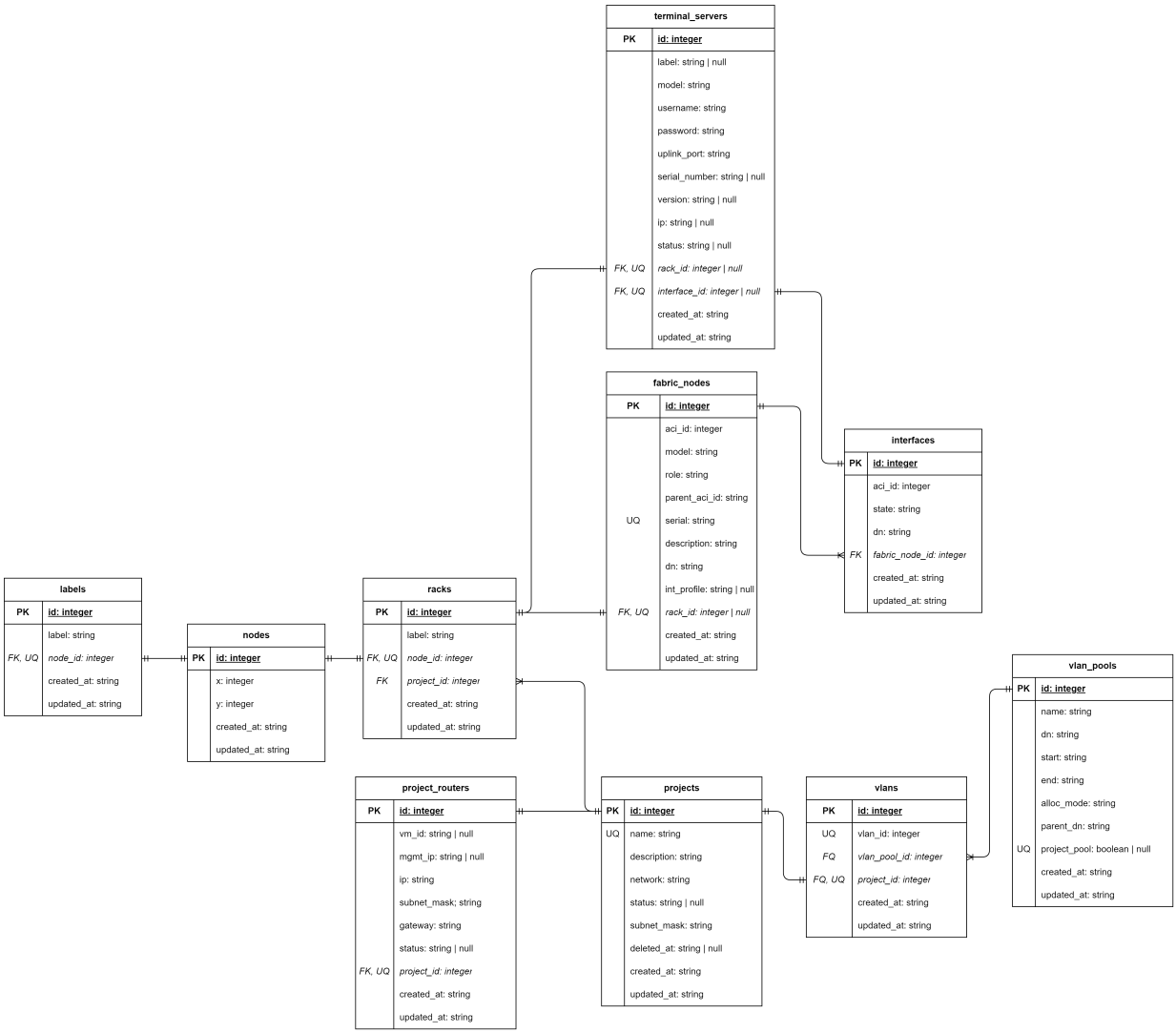

Figure 5.2: Database ERD

Nodes

Nodes provide the root of storing information about the layout of the rackspace. Each node can either have a rack or a label attached to it. Labels will serve as a place to store text information on the rackspace diagram for informational purposes only. Racks will represent a physical rack in the rackspace. By using a node table, the position of the node can be abstracted away from the other data which is more relevant to the automation functionality of the solution. A more refined and efficient query is also possible as all nodes can be easily retrieved by selecting all nodes and then joining the rack and label tables to retrieve the relevant information.

Fabric Nodes and Interfaces

The fabric nodes and interfaces tables will be used to store information related to all fabric nodes that are attached to ACI. They will be automatically populated by the automation script which will retrieve information about the nodes and their associated interfaces

from ACI. Each interface will tie to a fabric node so that all interfaces belonging to a fabric node can be easily retrieved. A role and parent ID field are also included in the fabric node table which allows for FEXs to also be stored as nodes. ACI treats FEXs as child nodes to leafs, hence why the parent ID and role field are needed to provide the correct differentiation between the two.

Terminal Servers

The terminal servers table will store information about the various terminal servers present in the rack space, these will be inserted manually via the web UI. A relation will also exist that associates an interface to a terminal server so that the automation scripts can appropriately configure the uplink ports on the ACI fabric.

Projects

The projects table will store all projects present in the application. Each project will link to the racks via a project ID field in the racks table. The project's private subnet will also be stored with the project.

Project Routers

A project router will have a one-to-one relationship with a project, allowing only one project router per project. This table will keep track of the virtual router that is created for each project, and will also store the WAN IP address assigned to the router.

VLANs and VLAN Pool

The VLAN pool table will store a list of all VLAN pools present in ACI. It will also store the selection that the user has made as to the VLAN pool that should be used by the automation platform for its endpoint groups. The VLANs table will be used to keep a record of which project is using which VLAN within a VLAN pool so that a VLAN cannot be used more than once.

5.2 Testbed

To support the development of the automation platform, it is necessary to have a network that can be used to test the automation functionality on real hardware and software that would be used in production. The network will be built using a scaled-down version of what could be deployed in a real scenario, the same also applies to the associated compute and storage resources.

5.2.1 OOB Network

To provide connectivity for management and day-zero configuration, it is important to have a reliable OOB network. The purpose of a OOB network is to provide access to networking and infrastructure that is external to the main network so that in the event of a failure, the devices can still be reached to rectify any problem that may have occurred. The

OOB network will be a single layer 2 network, with a single switch providing connectivity to all devices. The switch will be a Cisco Catalyst 2960-XR, and will be connected to external infrastructure which will host a VPN server and a NAT router so that devices connected to the OOB network also have access to the internet. The VMs hosted on the ESXi server are also detailed in the topology shown in figure 5.3

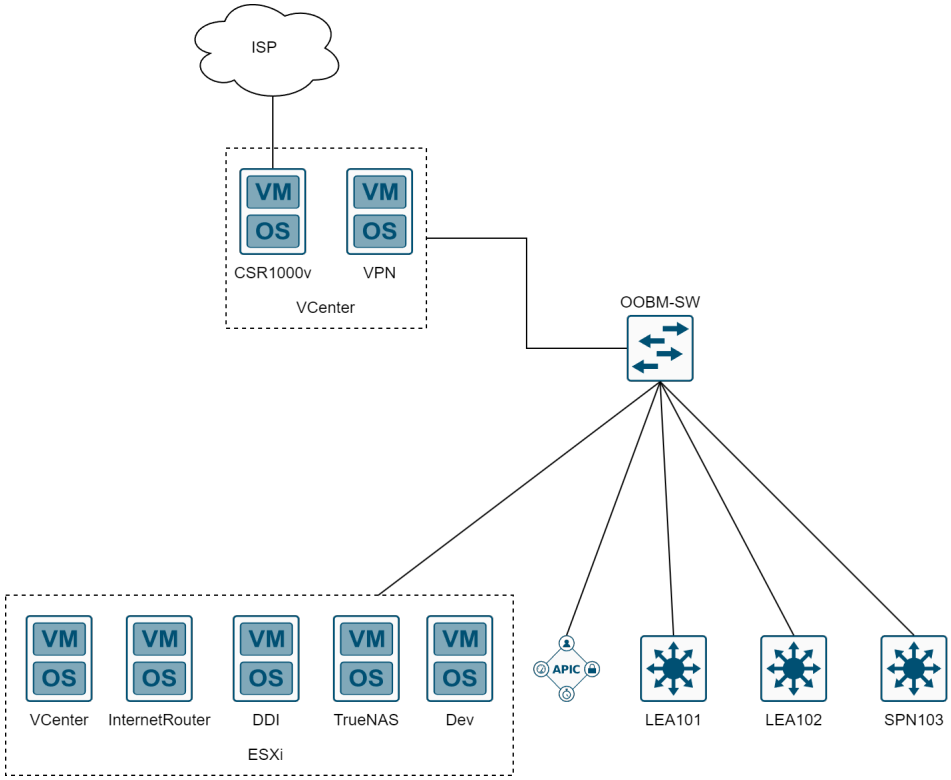

Figure 5.3: OOB Topology

5.2.2 ACI Fabric

As the automation platform is designed to automate ACI fabrics, a reduced ACI fabric will be deployed for development and testing. The base fabric will consist of one spine and two leafs, these being the N9K-C9336PQ and N9K-C93180YC-EX respectively. The leafs will also have a total of 3 FEXs attached to provide RJ45 connectivity to the fabric, and to also ensure that the automation platform will correctly support FEXs. The FEXs used will be the N2K-C2248TP-E-1GE. ACI also requires at least one APIC to function, so a single APIC M2 will be connected to the pair of leafs to provide overall administration and control over the fabric.

A single ESXi host will also be included in the network, which will be also dual-homed to both leafs to provide connectivity, and also test LACP functionality. The ESXi host will be used to host the automation platform and will also host the associated infrastructure required for the automation platform to function, such as vCenter.

Two routers simulating terminal servers will also be included in the design. Cisco routers can have additional modules inserted into them allowing them to provide console connectivity to devices via SSH or Telnet. Shown in figure 5.4 is the fabric topology that will be used for the testbed. Device names/hostnames are shown in the figure for clarity.

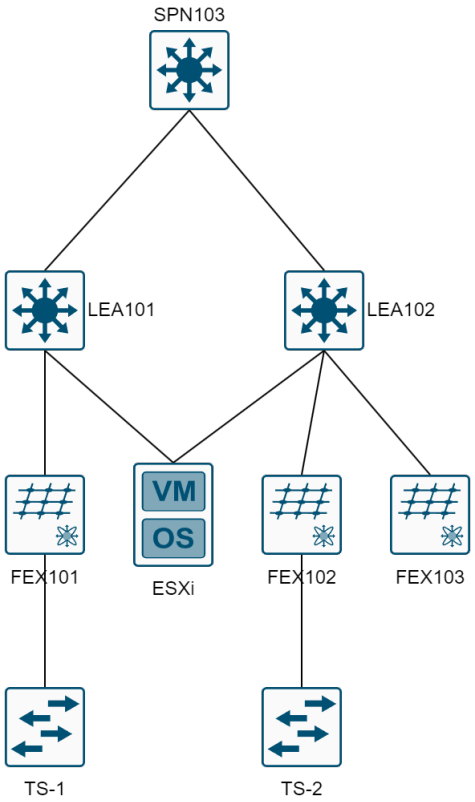

Figure 5.4: Fabric Topology

5.2.3 ACI Policy

To provide internal connectivity within the ACI testbed, the correct policy will need to be designed to facilitate this. As ACI is policy-based, conventional networking paradigms are shifted and abstracted behind ACI. Figure 5.5 shows the tenant and the policy it houses that will be created to provide connectivity within the testbed.

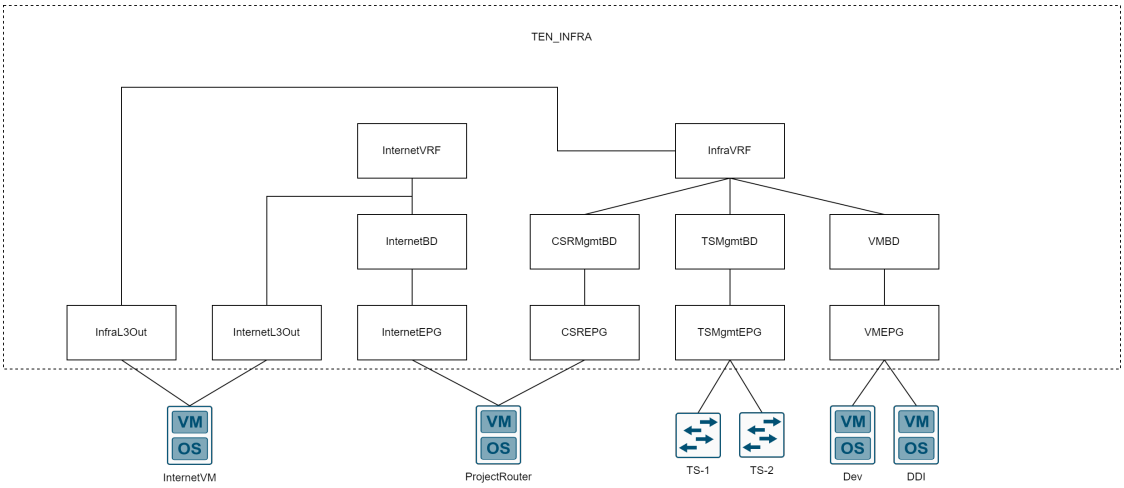

Figure 5.5: ACI Policy Overview

Infra VRF

The infra VRF will provide L3 routing ability between all associated bridge domains and EPGs for the sake of simplicity, all testbed connectivity for infrastructure will be able to communicate with one another.

InfraL3Out

The InfraL3Out will be used to advertise connectivity to outside the ACI fabric, the purpose of this is to provide internet connectivity via a NAT router as ACI can't provide NAT. This results in any packets destined for anything other than locally attached routes present in the InfraVRF will be routed out of the InfraL3Out.

OSPF will be used to share routes between ACI and the NAT router.

Infra Bridge Domains

Three bridge domains bring the ability to split connectivity into different subnets and broadcast domains. This allows for L2 functionality such as DHCP to be easily controlled and connectivity to be segmented. There will be a bridge domain to provide connectivity for the following:

- *Virtual Project Routers*
  - *DHCP relay will be operational to forward DHCP discover requests onto the DDI server attached to the VM bridge domain, thus allowing newly created VMs to receive an IP address automatically, to facilitate being provisioned by the automation platform.*
- *Terminal Servers*
  - *Allows the automation platform to reach the terminal servers via their RESTCONF API to apply the configuration.*

- *Infrastructure and testing VMs*
  - *Provides connectivity to the fabric for testing and provides services such as DHCP.*

Infra EPGs

Each EPG has a one-to-one relationship with a bridge domain and is used to provide connectivity via VMWare ACI integration and static port associations.

Internet VRF/BD/EPG

This VRF/BD/EPG will be used to emulate the project routers having a connection to the internet. This will be used to test the automation platform's ability to configure the project routers to have internet connectivity.

Connectivity will be provided via the InternetL3Out using OSPF for routing advertisements to the same NAT router that provides internet connectivity to the Infra VRF, although a different subnet will be used.

IP Addressing

Property	Network Address	Gateway
InternetBD	172.16.0.0/24	172.16.0.254
TSMgmtBD	172.16.1.0/24	172.16.1.254
CSRMgmtBD	172.16.2.0/24	172.16.2.254
VMBD	172.16.3.0/24	172.16.3.254
OOB	192.168.0.0/24	192.168.0.254

Table 5.1: Functional Requirements

Chapter 6

Implementation

This chapter will detail the development and implementation of the automation platform as well as the configuration of ACI and vCenter

6.1 Testbed Construction

To allow the solution to be compatible with an ACI fabric and vCenter environment, a testbed was required to develop and test the solution against. The design of this testbed was illustrated in chapter 5.2.2. Shown below in Figure 6.1 is the physical testbed deployment used for this project, as well as an outline of each device's name and function.

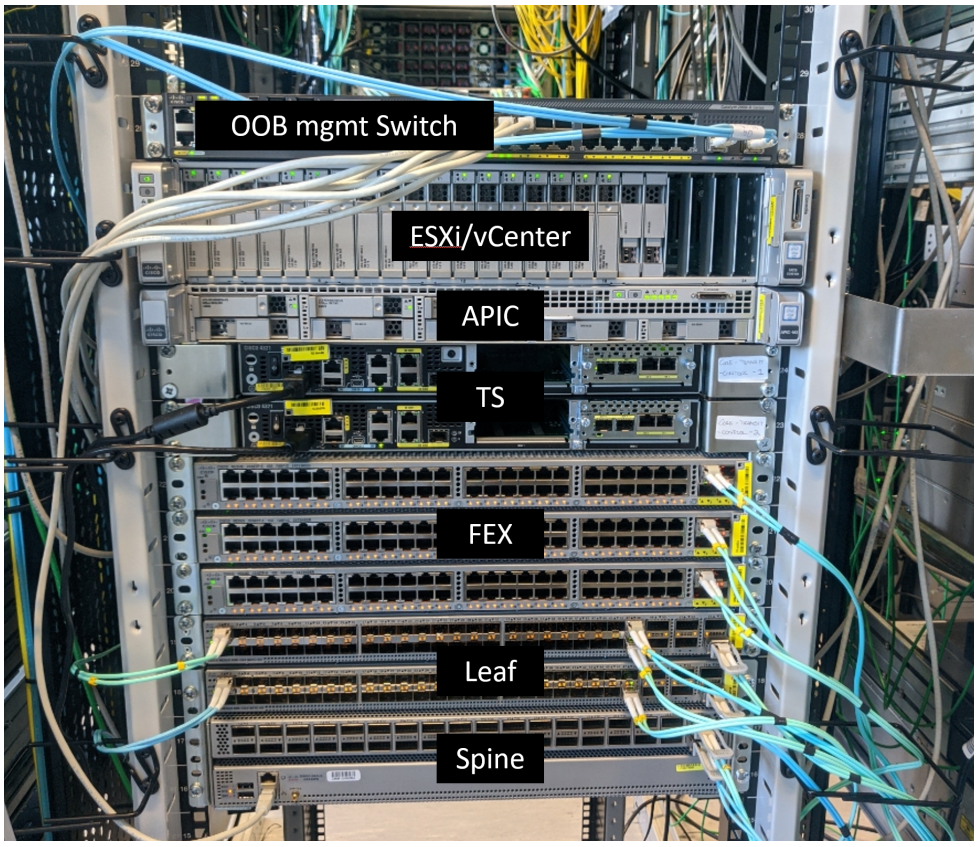

Figure 6.1: Testbed Physical Deployment

6.2 ACI Configuration

The initial wizard was used to set up the fabric and bring it up to a level where configuration can be applied. During the wizard, the first thing to configure is the fabric membership, where the auto-discovered spines and leafs are onboarded and registered into APIC. One of the most important settings is to configure the spine to perform as a BGP route reflector, this is because MP-BGP is used internally inside of ACI to advertise routes. Figure 6.2 shows how AS 65001 was chosen.

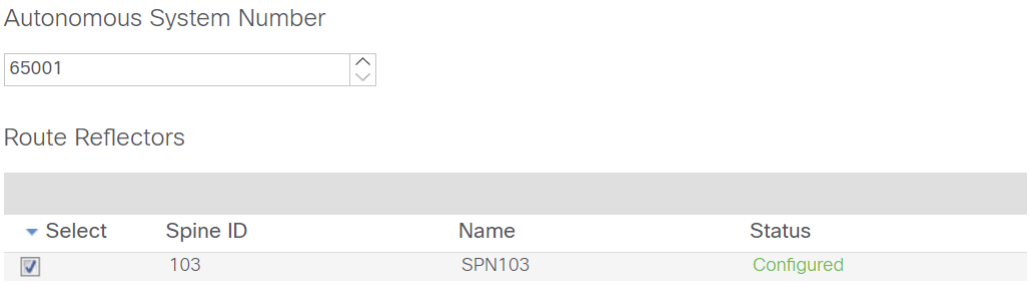

Figure 6.2: BGP Route Relfection

Other settings such as DNS and NTP were configured to ensure that all devices have name resolution and that all clocks are in sync. OOB IP addresses were also configured so that the fabric nodes can be reached via SSH for troubleshooting purposes, figure 6.3 shows the configuration of the OOB IP addresses.

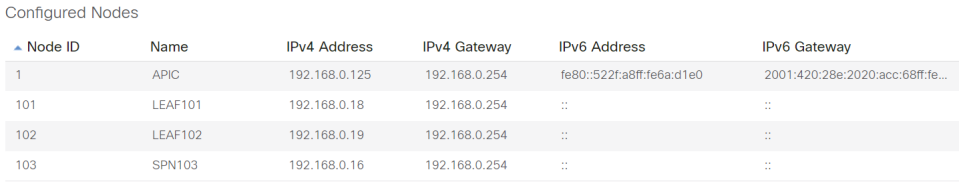

Figure 6.3: ACI Out of Band IP Configuration

6.2.1 VMware vCenter Integration with ACI

ACI provides the handy functionality of being able to integrate with vCenter and automatically push created EPGs into vCenter in the form of DPGs with the VLAN tagging being handled automatically. Firstly, a dynamic VLAN pool was created which is required so that VLANs can automatically be associated with EPGs and DPGs. Figure 6.4 shows the configuration of the VMWare integration.

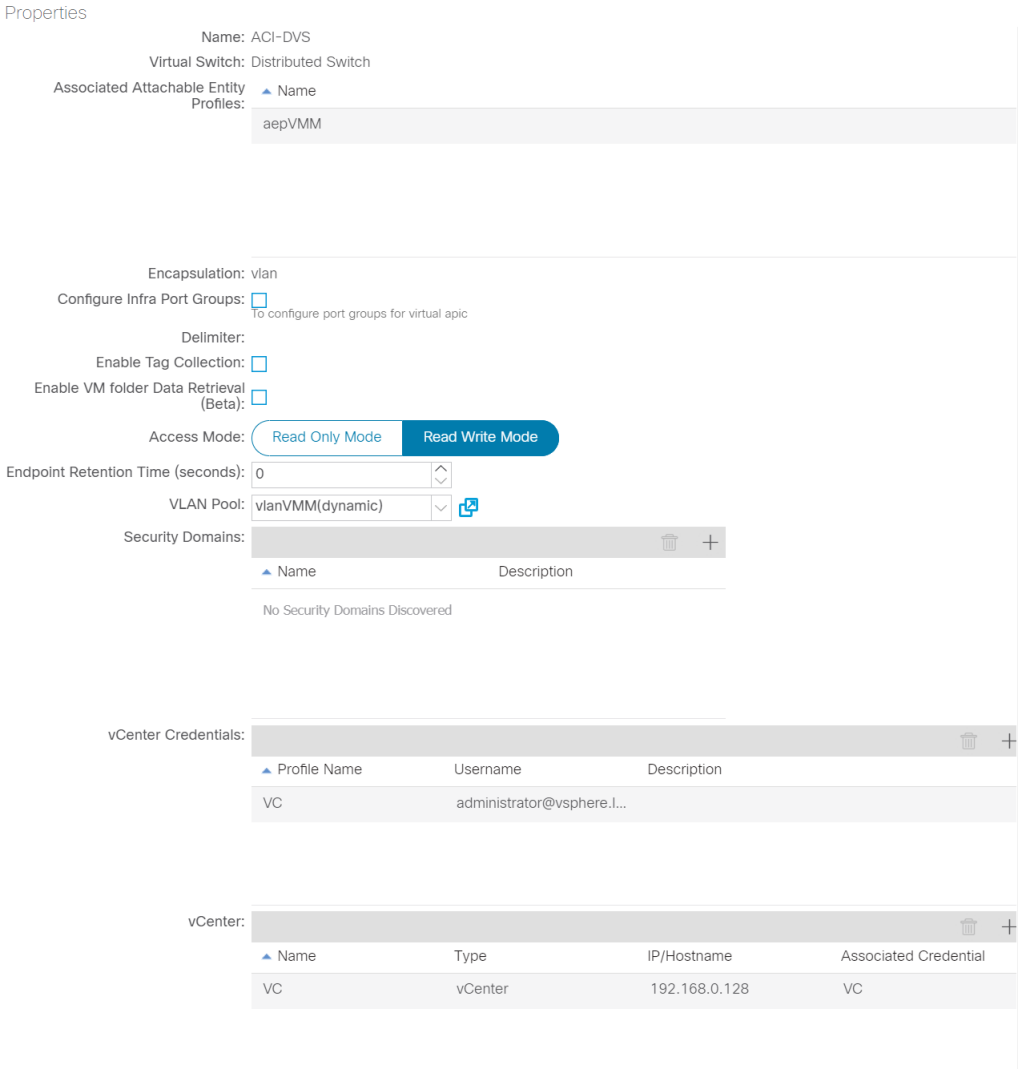

Figure 6.4: VMWare Integration Configuration

To get the dual-homed connection of the ESXi host to the two leafs operational, the two leafs must be brought together to form a vPC pair. Firstly, a vPC protection group must be created to inform ACI that these two leafs should have a keepalive link formed via the spine between them.

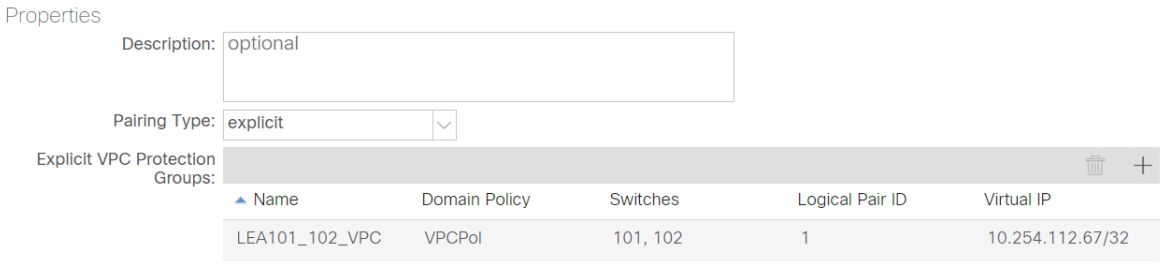

Figure 6.5: vPC Protection Group

A vPC policy group can then be created to associate the VMWare integration with the

AAEP, and to also configure the interfaces associated with the group to use LACP.

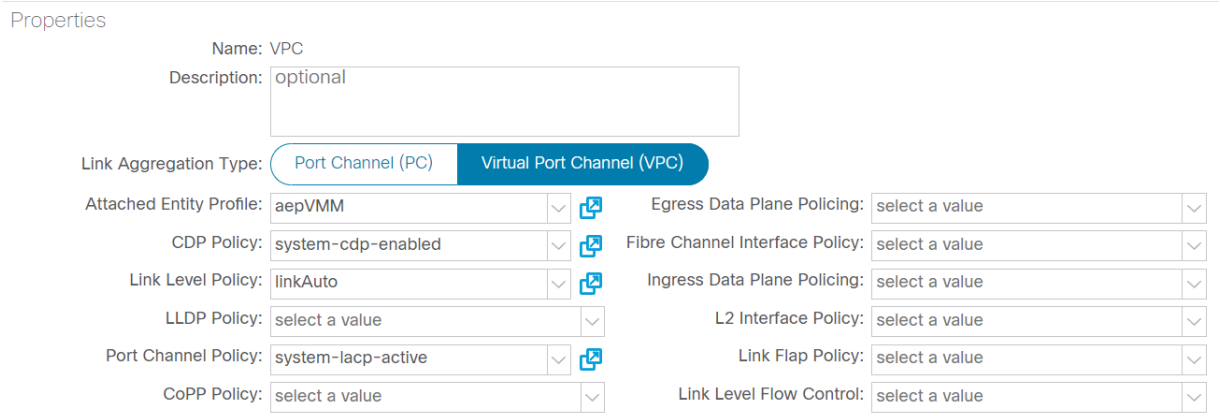

Figure 6.6: vPC Interface Policy Group

This vPC policy group can then be applied to the interfaces on the leafs via an interface profile which is shown in figure 6.7.

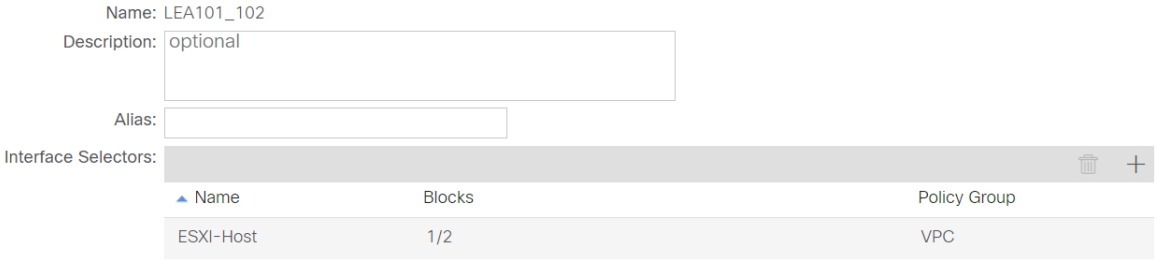

Figure 6.7: vPC Interface Assignment

In vCenter, figure 6.8 shows that ACI has correctly pushed the DVS to vCenter.

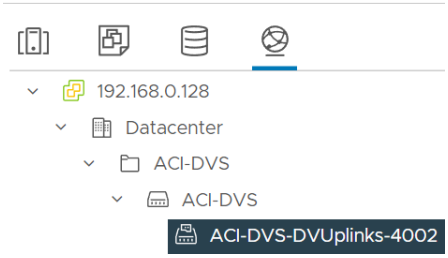

Figure 6.8: vCenter Distributed Virtual Switch

Now the uplinks need to be assigned to the LACP group that has been created by ACI. Figure 6.9 shows the configuration of the uplinks to use LACP.

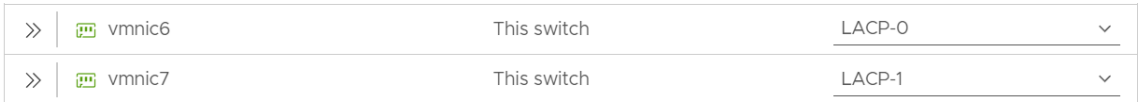

Figure 6.9: Uplink LACP Configuration

The command `fab 101 show vpc extended` *can then be executed on the APIC via SSH to retrieve the status of all vPC present on leaf 101.*

```
    vPC status
----------------------------------------------------------------------------
id   Port   Status Consistency Reason      Active vlans         Bndl Grp Name
--   ----   ------ ----------- ------      -------------------- -------------
343  Po3    up     success     success     1135,1168-1169,1200- VPC
                                            1201
```

Figure 6.10: vPC Status on Leaf 101

6.2.2 TEN_INFRA Tenant Setup

With vCenter integration complete, the tenant can now be created that will house all policies related to the infrastructure of the testbed. All of the VRFs/BDs/EPGs were created in accordance with the topology shown in figure 5.5.

To get external connectivity working with the NAT router that will be provisioned inside vCenter, L3Out configuration must be created. Figure 6.11 shows the configuration of the L3Out that will be used for the Infra VRF, an OSPF area of 1 has been specified. A floating SVI has been utilised so that the VMWare integration configured earlier can be used to pass the L3out EPG to vCenter automatically. A new static VLAN pool was created along with a new L3 domain. This L3 domain was then added to the VMWare integration so that L3Outs can then use the VMWare virtual domain.

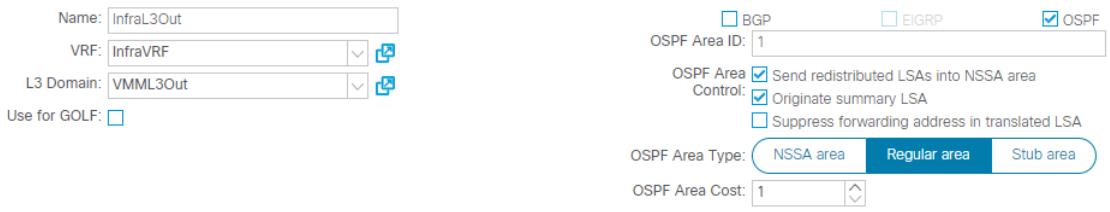

Figure 6.11: Infra L3Out Configuration

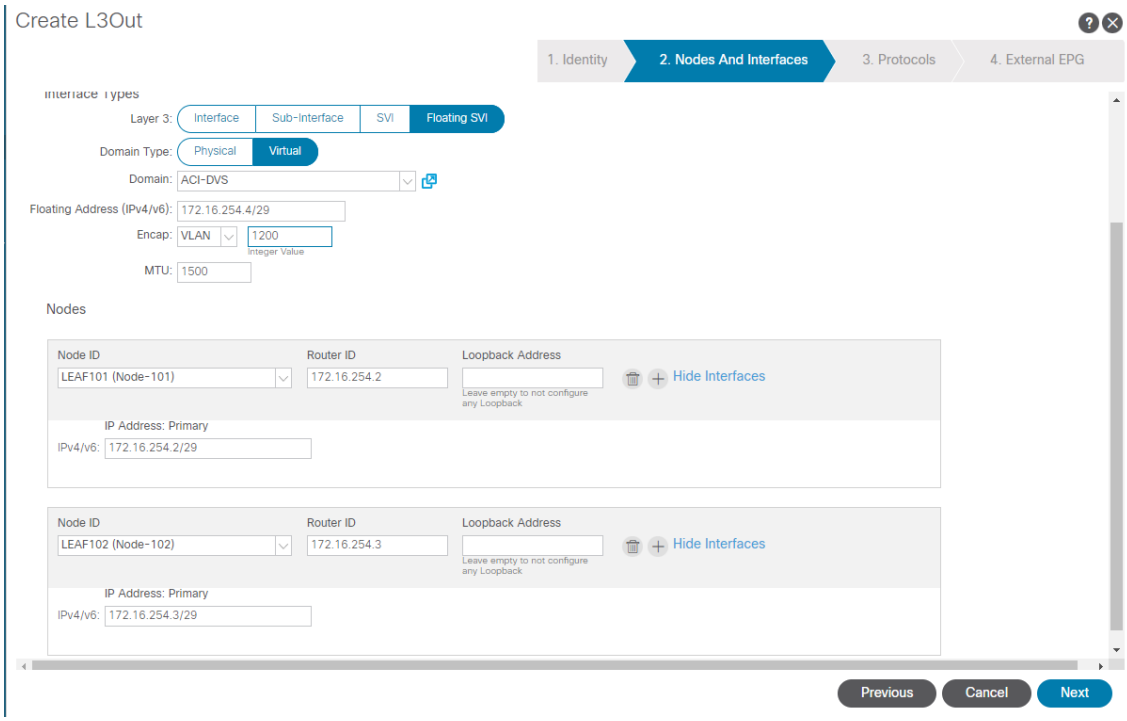

Figure 6.12: Adding the leafs to the L3Out

It is then important to assign the L3out interface profiles to the enhanced LACP group so that the OSPF and traffic flow utilise LACP to vCenter correctly. Figure 6.13 shows the configuration of the L3Out interface profiles.

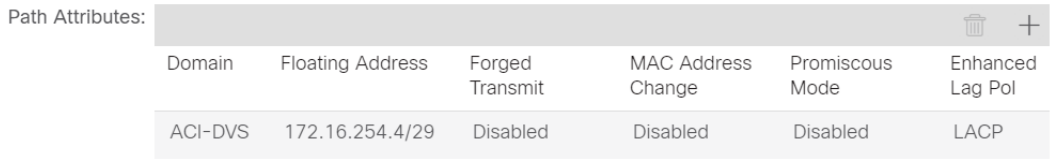

Figure 6.13: L3Out Interface Profile Configuration

The same process will be repeated for the InternetL3Out for the InternetVRF.

The NAT router can then be deployed and attached to the L3Out EPGs via the DPGs that have been pushed to vCenter by ACI.

NAT Router Configuration

For virtual routing, the Cisco CSR1000v Virtual Router was chosen. The interfaces were configured with IP addresses that are present in the subnets that were used in the earlier L3Out configuration. OSPF was also configured so that the routes to and from ACI are advertised correctly. The following OSPF configuration was used for the NAT router.

```
router ospf 1
passive-interface default
no passive-interface GigabitEthernet1
network 172.16.254.0 0.0.0.7 area 1
default-information originate
!
router ospf 2
passive-interface default
no passive-interface GigabitEthernet2
network 172.16.254.8 0.0.0.7 area 2
default-information originate
```

Figure 6.14: NAT Router OSPF Configuration

To verify the OSPF configuration is working correctly, the following command and output was executed on the NAT router.

```
InternetRouter#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
172.16.254.10     1   FULL/BDR        00:00:35    172.16.254.10   GigabitEthernet2
172.16.254.11     1   FULL/DR         00:00:33    172.16.254.11   GigabitEthernet2
172.16.254.2      1   FULL/BDR        00:00:39    172.16.254.2    GigabitEthernet1
172.16.254.3      1   FULL/DR         00:00:32    172.16.254.3    GigabitEthernet1
```

Figure 6.15: NAT Router OSPF Neighbors

Basic NAT overload can then be configured for all subnets present on ACI so that the various bridge domains now have internet via the router.

6.3 vCenter Configuration

As ACI has configured the virtual switch and port groups for us, the only configuration required is to make a VM that will serve as a project router template that the automation scripts will then clone. Due to a limitation with the vCenter REST API, the VM to be cloned will be a regular VM and not a template. This is because the REST API does not support cloning templates.

The router will have 3 ethernet interfaces:

- *GigabitEthernet 0 - Project Network*

  - *IP Address set by automation platform.*

- *GigabitEthernet 1 - WAN Uplink*
  - *IP Address set by automation platform.*
- *GigabitEthernet 2 - Management*
  - *IP Address obtained by DHCP so the automation platform can connect.*

The router will need to obtain an IP address from DHCP on the CSRMgmt EPG so that the automation platform can connect via RESTCONF. The router will also have an interface in the Internet EPG and then the other interface will be set to the quarantine DPG so that it can be assigned to the EPG created for the project by the automation platform. NAT overload can also be preconfigured on the router so that the project can access the internet, however, the NAT ACL will need to be configured via RESTCONF by the automation platform.

6.4 Automation Platform

Both backend and frontend features were developed simultaneously as that was the most efficient way of development. Instead of backend and frontend implementation detailed separately, development of features will instead be covered.

6.4.1 Packages

To accelerate development, many packages and libraries were used.

Next.js

Next.js is based on React and is a framework for server-side rendering and static site generation. It is used to create the frontend of the automation platform. It features many benefits over using plain React, such as the ability to pre-render many parts of the web application resulting in a reduction of load times upon page load. TypeScript is also supported which was used to aid development by enforcing the usage of types when defining variables and functions.

Tailwind CSS

Tailwind CSS is a utility-first CSS framework. It is used to style the frontend of the automation platform. It has many benefits over plain CSS, such as the ability to rapidly create a responsive UI through the use of pre-defined classes. The Flowbite React package was also utilised, which is a library of components that utilise Tailwind CSS for styling. Tailwind also features dark mode support which was used to create a dark theme for the automation platform.

React Flow

React Flow is a library that provides the ability to create and programmatically define flowcharts within React. It is used to create the rackspace layout feature and is the core of the automation platform. By using React Flow, development time was greatly reduced. It also has features such as the ability to define custom nodes to be displayed, which is required to get the desired look, feel and functionality for the application.

Laravel

Laravel is a PHP framework that makes it quick and easy to create feature-rich APIs which is why it was chosen. It features an easy-to-use routing system for easily routing API requests to the correct controller classes. It also makes use of the Eloquent ORM which makes it easy to create and query the database. The inbuilt Guzzle HTTP client also makes interacting with the various APIs required to build the automation scripts easy and simple.

6.4.2 Recreation of Rackspace

To allow for the recreation of the rackspace in a map-like user interface, React Flow was used with custom nodes that represent racks. To save the positions of the racks in 2D space, the following API endpoints were created: As React Flow uses the term 'node'

API Path	Method	Description
`/node/{id}`	`GET`	Get a node by ID
`/node/{id}`	`PATCH`	Update a node by ID
`/node/{id}`	`DELETE`	Delete a node by ID
`/node`	`GET`	Get all nodes
`/node`	`POST`	Create a new node

Table 6.1: API Routing Table

to refer to the individual elements in the flowchart, the API endpoints were also named 'node' to avoid confusion. Every time a node is moved on the flowchart, the API is called to update the position of the node in the database.

The ability to add labels to the rackspace representation was also deemed an important feature, as it would allow for useful pieces of information to be placed on the map. To facilitate this, a new custom node was created that just displays text. Shown below in figure 6.16 is the rackspace layout feature with a label used to illustrate the name of the row.

Figure 6.16: Rackspace Layout

React Flow provides a handy interface for allowing custom nodes to be dragged and dropped, allowing for a user-friendly way to add racks and labels to the space. Shown in figure 6.17 is the utility panel where racks and labels can be dragged and dropped onto the rackspace representation.

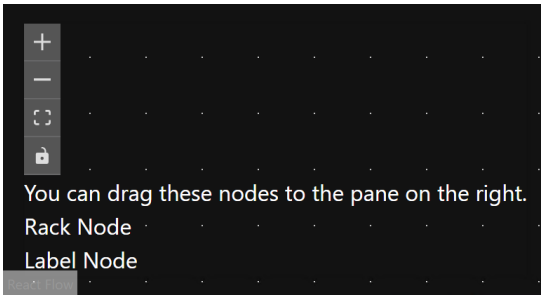

Figure 6.17: Drag and Drop

6.4.3 Initial ACI Integration

Fabric Nodes

With the ability to add racks to the space, the next feature to be developed was the integration with ACI. As the premise of the application is that each rack has a top of rack switch, the automation platform needs to be able to pull all of the available leafs and FEXs from the ACI fabric so that they can then be assigned to a rack. Because ACI treats a FEX as an extension of a leaf node, some logic is required to differentiate between the two. To retrieve FEXs, a list of all leaf nodes can be retrieved from ACI, the list of leaf nodes can then be iterated over and the following API endpoint can be used to retrieve a list of all FEXs attached to the leaf node:

`https://apic-ip/api/node/class/topology/pod-1/node-{leafID}/eqptExtCh.json`

The returned objects are FEXs that are attached to the leaf with the given ID. This information can be stored in the database with the `role` *column used to differentiate between a leaf and FEX. The column* `aci_parent_id` *is also set to the parent leaf as that will be required for subsequent API calls.*

With the nodes synced to the database, the next step is to provide the user to specify which interface profiles should be used by the automation platform to select the access ports on the leafs and FEXs. A requirement of the platform will be that an interface profile that belongs to a single node will have to be manually created as it is not easy to automate the creation of this. Two API queries will have to be made as FEX and leaf interface profiles are treated separately. The following API endpoints were used to retrieve this information:

```
https://apic-ip/api/node/mo/uni/infra.json?query-
target=subtree&target-subtree-class=infraAccPortP

https://apic-ip/api/node/mo/uni/infra.json?query-
target=subtree&target-subtree-class=infraFexP
```

A basic modal can then be created with a dynamic form that will allow the user to map the fabric nodes to an interface profile. Figure 6.18 shows the modal with the interface profile mapping form.

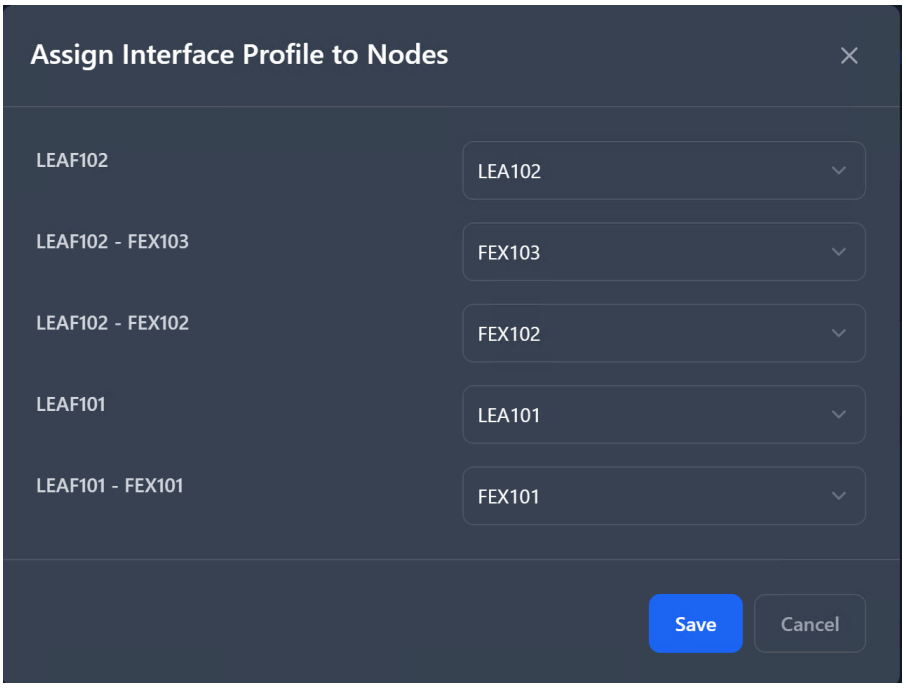

Figure 6.18: Interface Profile Mapping

The distinguished name of the interface profile selected can then be stored in the corresponding `int_profile` *column of the fabric node.*

VLAN Pools

So that the automation platform can assign unique VLANs to projects, the correct VLAN pool from ACI must be used so that the physical domain that the automation platform will

create is attached to the correct VLAN pool. The list of valid VLAN pools will then be presented to the user in the form of a dropdown menu where the desired VLAN pool can be set. Figure 6.19 shows the selection dropdown.

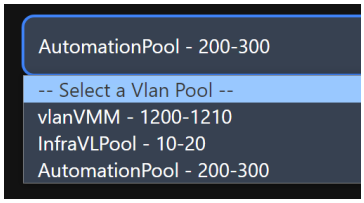

Figure 6.19: VLAN Pool Selection

The selected VLAN pool can then be saved in the database using the `project_pool` *column set to* `true` *for the corresponding pool. The future project logic will then be able to retrieve the start and ending VLAN that can be used and allocate individual VLANs to projects.*

6.4.4 Terminal Servers

Terminal servers will have a dedicated management page so that the status of connected terminal servers can be monitored. Cisco IOS-XE has an inbuilt RESTCONF server which will be used to retrieve and push configuration to the devices. A form modal to consume data was created which is shown in figure 6.20.

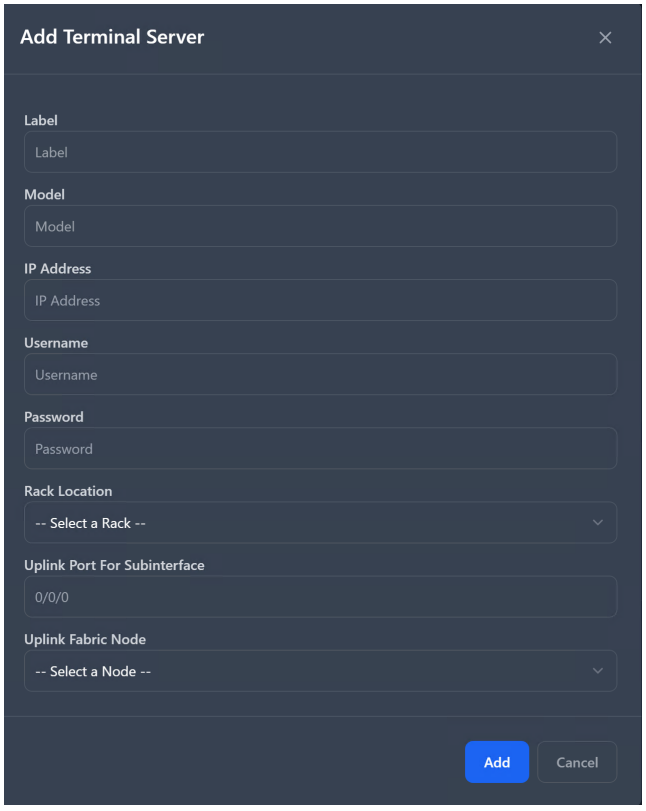

Figure 6.20: Terminal Server Addition Form

The notable features are the ability to assign the newly created terminal server straight to a rack, although that feature will also be added to the rackspace view. The port that is connected to the fabric from the terminal server will need to be manually provided, as the automation scripts will need to create sub-interfaces on top of that interface to facilitate communication to the project network. The port on the side of that link, the fabric side, will also need to be specified so that the automation scripts can create a trunk interface. The username and password to gain access to the terminal server are also required to be entered by the user.

6.4.5 Rack ACI Integration

As the premise of the automation platform is the ability to automatically include racks into a project's network, the platform needs to be aware of which fabric nodes belong to which rack. To achieve this easily, it was determined that the best method would be to provide an 'edit rack' modal that can be accessed by hovering over a rack. Figure 6.21 shows the popover that appears when hovering over a rack.

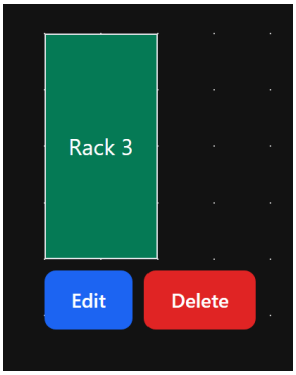

Figure 6.21: Edit Rack Popover

The ability to change a rack's name, as well as associate a fabric node and terminal server was provided in the edit modal, which is shown in figure 6.22.

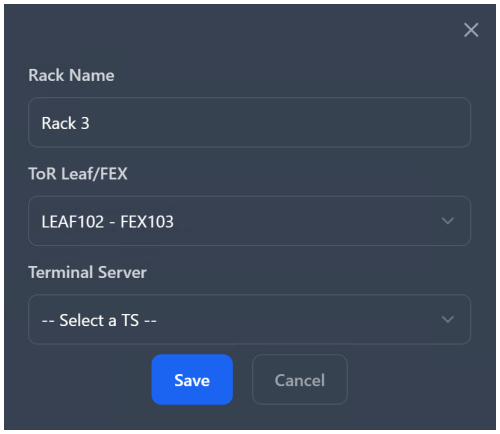

Figure 6.22: Edit Rack Modal

The logic to only allow a single fabric node and terminal server to a single rack was also incorporated to prevent user error. The `rack_id` *column of the fabric node and terminal server tables were updated to reflect the rack that they were assigned to.*

6.4.6 Project Creation

The final user-facing feature to be developed is the ability to create projects, which is the main feature of the automation platform. The user will have the ability to create and edit projects as well as delete them. The edit functionality will include adding and removing racks from the project, to facilitate project expansion and contraction which is a very common requirement.

A modal was decided upon as being the most user-friendly way of adding and editing projects. A multi-stage form that guides the user through the process of onboarding the project was also decided upon. The flow will be as follows:

1. *The user will be presented with a form to enter the project name and description.*
2. *The user will be presented with a list of racks that are available to be added to the project.*
3. *The user will be asked to specify a subnet to be used in the project network along with a WAN IP address. If no project subnet is specified, the next available subnet will be automatically assigned.*

Shown in figure 6.23 is the modal form section that allows the user to select the racks that the project will occupy.

Figure 6.23: Add Project Racks

Because of the componentised nature of React, the rackspace layout can just be imported again instead of redefining all code and properties. The racks shown in red are occupied by another project, and thus cannot be included in a newly created project. The form for specifying IP addressing is shown below.

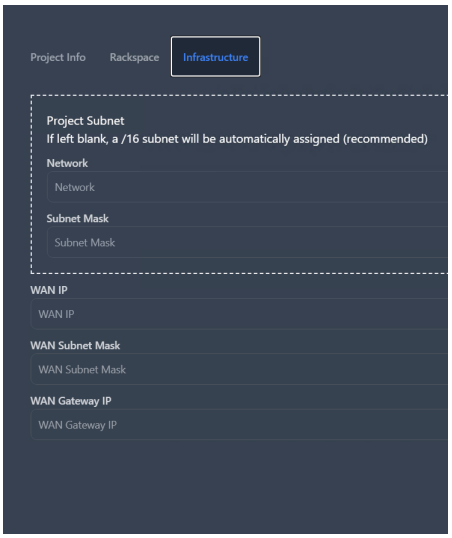

Figure 6.24: Specifying project IP addressing

The same modal layout is used for editing a project, however, the logic is revised to allow racks to be deselected and removed from the project, at the same time as allowing new racks to be added.

6.4.7 Project Automation

With the ability to create projects, the automation platform needs to be able to automatically create the project's network infrastructure. To perform this, Laravel Queues were decided upon, as they allow tasks and API queries to be performed outside of the process that responds to the user's request. This allows the user to continue using the platform while the automation scripts are running in the background.

ACI Fabric

The first step when provisioning a project is to configure the ACI fabric, as that will be required before any VMs can be created and attached to the project network.

```
{
    $aciClient = new ACIClient();
    $vmWare = new vSphereClient();
    $project = Project::find($this->projectId);
    if ($aciClient->createTenant($this->projectName)) {
        if ($aciClient->createBD($this->projectName)) {
            if ($aciClient->createAP($this->projectName)) {
                if ($aciClient->createEPG($this->projectName)) {
                    if ($aciClient->associatePhysDom($this->projectName)) {
                        if ($aciClient->deployToNodes($this->projectId)) {
                            $project->status = 'VMware';
                            $project->save();
                            if ($vmWare->deployProjectRouter($this->projectName,
                            $this->projectId)) {
                                VirtualRouterProvision::dispatch($this->projectId)
                                ->delay(Carbon::now()->addSeconds(140));

                                TSProvision::dispatch($this->projectId);
                                return true;
                            }
                        }
                    }
                }
            }
        }
    }
    $project->status = 'Error';
    $project->save();
    return false;
}
```

Figure 6.25: Project Creation Job

Figure 6.25 shows the job that is dispatched when a project is created. The job performs the following actions:

1. *Creates the tenant in ACI where all network configurations related to the project will be stored. The project VRF will automatically be created with this API call.*
2. *Creates the bridge domain to allow L2 communication between attached ports and VMs.*
3. *Creates the application profile that will house the EPG.*
4. *Creates the EPG that will be used to attach static ports and VMs to the project network.*
5. *Associates the created EPG with the Automation and Terminal Server physical domains. It also associates with the VMware integration domain so that the created EPG is automatically extended into VMware vCenter.*
6. *The fabric node and their ports that are attached to the member racks that the*

project has been allocated to are then added into the EPGs static mapping.

7. *The project router is then deployed through the use of vCenters ability to clone a VM using the API. When the VM has been cloned, a new job is dispatched with an initial delay of 140 seconds. This job will find the newly created VMs IP address via VMWare guest tools and then configure the VM with the appropriate network configuration. The delay is added to give the VM time to boot up and acquire an IP from DHCP.*

8. *A job to configure the terminal servers is also dispatched.*

Virtual Router Provisioning

Shown in figure 6.26 is the job that is dispatched from the create project job, which will in turn provision the newly cloned project router.

```
{
$vmWare = new vSphereClient();
$project = Project::with('projectRouter')->find($this->projectId);
for ($i = 0; $i < 10; $i++) {
    $routerIp = $vmWare->getVmIp($project->projectRouter->vm_id);
    if ($routerIp !== false && $routerIp != '0.0.0.0') {
        $httpClient = new IOSXEClient($routerIp);
        if ($httpClient->connectionTest()) {
            if ($httpClient->setHostname($project->name . '-CSR') &&
                $httpClient->setAddresses($project->projectRouter->ip,
                $project->projectRouter->subnet_mask, $project->network,
                $project->subnet_mask, $project->projectRouter->gateway)) {

                $httpClient->save();
                $project->status = 'Provisioned';
                $project->save();
                return true;
            } else {
                $project->status = 'Error';
                $project->save();
                return false;
            }
        } else {
            sleep(10);
        }
    } else {
        sleep(10);
    }
}
return false;
}
```

Figure 6.26: Project Router Configuration Job

The first step the algorithm takes is to enter a for loop that will iterate for a maximum of 10 times. The vCenter API will be contacted to retrieve a list of IP addresses belonging

to a specific VM. As the ID of each project router associated with a project is stored in the database, this ID can be used to get the IP address of a project's router. If the VM has not yet received an IP address, then the loop will sleep for 10 seconds and then try again. Once an IP address has been received and registered by vCenter, the next step can be performed. The IP address is then used to create a new instance of the IOS-XE client, which is used to configure the router via RESTCONF. The hostname and addresses are pulled from the database and pushed to the router. Other settings such as ACLs are also generated from the addresses provided. Once the router has been configured, the project status is updated to 'Provisioned' and the project is ready to be used.

Terminal Server Provisioning

The terminal server provisioning job is shown in figure 6.27. This job is dispatched from the create project job, and will configure the terminal servers that are attached to the project's racks.

```
{
    $project = Project::with('racks.terminalServer', 'vlan')->find($this->projectId);
    foreach ($project->racks as $key => $rack) {
        if ($rack->terminalServer !== null) {
            $iosXE = new IOSXEClient($rack->terminalServer->ip,
            $rack->terminalServer->username, $rack->terminalServer->password);
            if ($iosXE->connectionTest()) {
                if ($iosXE->setSubInterface($this->firstUsableIP($project->network,
                  $project->subnet_mask, $key), $project->subnet_mask,
                  $project->vlan->vlan_id, $rack->terminalServer->uplink_port)) {
                    $iosXE->save($rack->terminalServer->username,
                      $rack->terminalServer->password);
                    return true;
                }
            }
        }
    }
    return false;
}
```

Figure 6.27: Terminal Server Configuration Job

The algorithm iterates through all of the terminal servers that belong to the project via the relation that a terminal server belongs to a rack. The first step is to check if the rack has a terminal server associated with it, if it does then the algorithm can proceed. A sub-interface on the terminal server is then configured with an appropriate IP address, which is generated from the first address in the subnet and up. The project VLAN is also included so that the correct encapsulation can be set on the sub-interface. The sub-interface is then saved to the terminal server. The algorithm then returns true to indicate that the job has been completed successfully.

Chapter 7

Testing

To ensure that the platform works correctly and doesn't break any connectivity outside the remit of the automation platform, several tests were devised. Virtual Machines attached directly to hardware NICs were used. These VMs were connected as follows:

Test VM	Fabric Port
1	FEX101/1
2	FEX102/1
3	FEX103/1

Table 7.1: Test VM fabric connections

This will allow for the testing of communication between different fabric nodes to determine if the ACI fabric has been provisioned correctly. Access to the internet from the test VMs will also be tested to ensure that the virtual router has been provisioned correctly. Connectivity to the project's terminal servers will also be tested from the test VMs. The automation platform was setup as follows:

Rack	Fabric Node	Terminal Server
1	FEX101	TS-1
2	FEX102	TS-2
3	FEX103	

Table 7.2: Rack to Fabric Node and Terminal Server mapping

7.1 Project Communication

A new project was created with racks 1 and 2 being selected as members. Figure 7.1 shows the resulting project.

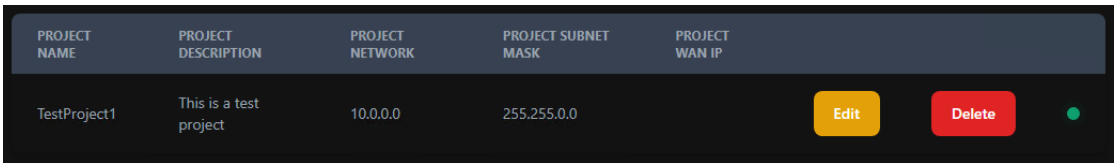

Figure 7.1: Test Project 1 created successfully

As can be seen, the project has been allocated a subnet of 10.0.0.0/16. This can be used to set IP addresses on the two test VMs. The IP addresses of 10.0.1.1 and 10.0.1.2 were chosen respectively. After assigning the IP addresses, the test VMs were able to ping each other. Figures 7.2 and 7.3 show the successful ping test, showing that the ACI fabric is being provisioned correctly.

```
→  ~ ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.618 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.463 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.434 ms
```

Figure 7.2: Test VM 1 pinging Test VM 2

```
→  ~ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=64 time=0.383 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=64 time=0.518 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=64 time=0.423 ms
```

Figure 7.3: Test VM 2 pinging Test VM 1

The next test was to ping the internet from the test VMs. This was done by pinging the IP address of both Cloudflare and Google DNS, which is shown in figure 7.4. This shows that the virtual router is being provisioned correctly.

```
→  ~ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=56 time=7.29 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=56 time=3.50 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=56 time=3.16 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 3.160/4.648/7.291/1.873 ms
→  ~ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=3.71 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=3.54 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=118 time=3.82 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 3.538/3.688/3.822/0.116 ms
```

Figure 7.4: Test VM 1 pinging the internet

7.2 Terminal Server Reachability

The terminal servers are automatically allocated the first available IP addresses sequentially in the order of the racks. In the case of this test project which has been allocated the network of 10.0.0.0/16, the addresses of the terminal servers will be 10.0.0.1 and 10.0.0.2 respectively. When pinging, 10.0.0.1 responded, however, TS-2 - 10.0.0.2 did not. Upon checking the configuration of TS-2, it was noted that no sub-interface configuration had been applied to the device. Figure 7.5 shows the problematic configuration script.

```
$project = Project::with('racks.terminalServer', 'vlan')->find($this->projectId);
foreach ($project->racks as $key => $rack) {
    if ($rack->terminalServer !== null) {
        $iosXE = new IOSXEClient($rack->terminalServer->ip,
          $rack->terminalServer->username, $rack->terminalServer->password);
        if ($iosXE->connectionTest()) {
            if ($iosXE->setSubInterface($this->firstUsableIP($project->network,
              $project->subnet_mask, $key), $project->subnet_mask,
               $project->vlan->vlan_id, $rack->terminalServer->uplink_port)) {
                $iosXE->save($rack->terminalServer->username,
                  $rack->terminalServer->password);

                return true;
            }
        }
    }
}
return false;
```

Figure 7.5: Terminal Server Provisioning Script

The issue is that the return statement will break the execution after the successful provisioning of the first terminal server, hence only the first was being provisioned. The fix was to remove the return statement to after the foreach loop. Figure 7.6 shows the connectivity which was successful after the bug fix.

```
➜  ~ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=0.378 ms
^C
--- 10.0.0.1 ping statistics ---
2 packets transmitted, 1 received, 50% packet loss, time 1030ms
rtt min/avg/max/mdev = 0.378/0.378/0.378/0.000 ms
➜  ~ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=2 ttl=255 time=0.355 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 1 received, 50% packet loss, time 1030ms
rtt min/avg/max/mdev = 0.355/0.355/0.355/0.000 ms
```

Figure 7.6: Test VM 1 pinging TS-1 and 2

7.3 Multiple Projects

To test intra-project communication is not possible, so the original test project was reduced to occupying one rack. A new project was created, consuming the remaining two racks. This will test to ensure that the automation platform correctly provisions the fabric and terminal servers for each project and that the automation script correctly deallocates and allocates racks, even with existing projects. Figures 7.7 and 7.8 show the new project, which has been successfully added to the platform.

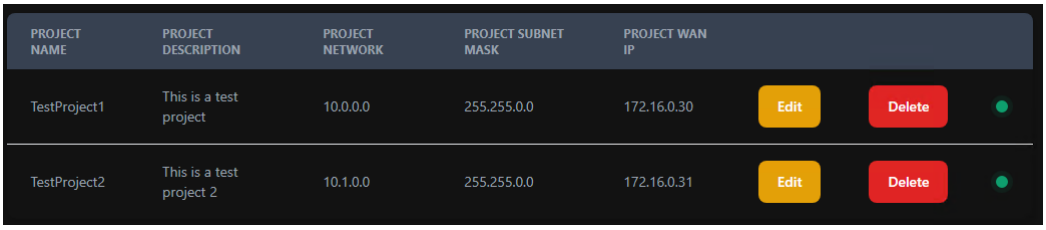

Figure 7.7: Test Project 2 created successfully

Figure 7.8: Rack Allocation with additional project

Test VMs 1 and 2 were reconfigured with the IP addresses of 10.1.1.1 and 10.1.1.2 respectively to account for the change in the subnet. A ping test was performed between the two VMs, which was successful. Figure 7.9 shows the successful ping test.

```
→  ~ ping 10.1.1.2
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.368 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.459 ms
^C
--- 10.1.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1023ms
rtt min/avg/max/mdev = 0.368/0.413/0.459/0.045 ms
```

Figure 7.9: Test VM 3 pinging Test VM 2

The reachability of TS-2, which is now in the second project was also tested, via SSH from Test VM 3. This was successful, as shown in figure 7.10. This shows that the automation platform correctly allocates and deallocates racks, even with existing projects.

```
→  ~ ssh admin@10.1.0.1
The authenticity of host '10.1.0.1 (10.1.0.1)' can't be established.
RSA key fingerprint is SHA256:otgeW9kLrvVrks0249aGRau+kWMxEgd2A+7932ne7cc.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.1.0.1' (RSA) to the list of known hosts.
Password:


TS-2#
```

Figure 7.10: Test VM 3 pinging TS-2

To verify the fact that the projects do not have any communication between them, the IP address of Test VM 2 was reverted to the 10.0.0.0/16 subnet, and 10.0.1.1 was pinged, which is the IP of test VM 1. As seen in figure 7.11 the ping test has failed, verifying that the projects are isolated from one another.

```
→  ~ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Destination Host Unreachable
From 10.0.1.2 icmp_seq=2 Destination Host Unreachable
From 10.0.1.2 icmp_seq=3 Destination Host Unreachable
^C
--- 10.0.1.1 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4091ms
pipe 4
```

Figure 7.11: Test VM 2 pinging Test VM 1 from different projects

7.4 Project Deletion

To verify that the correct fabric policies, terminal server interfaces and project router VM are correctly deleted, both projects were deleted. The configuration of the ACI fabric was then inspected along with vCenter inventory. Figure 7.12 shows that the tenants have been deleted automatically. Figure 7.13 shows that the interface policies have been deleted correctly. Figure 7.14 shows the project VMs have been cleared up appropriately.

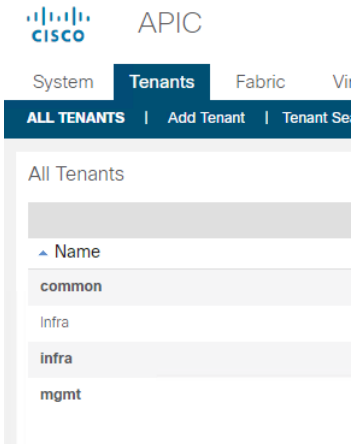

Figure 7.12: ACI Fabric Tenants

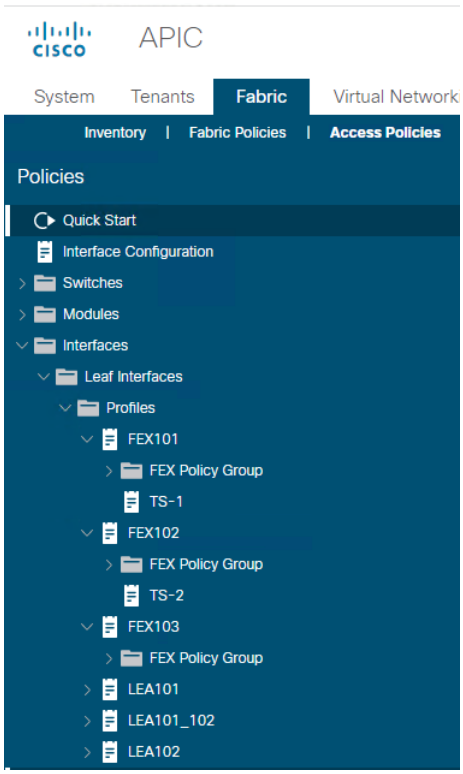

Figure 7.13: ACI Fabric Interface Policies

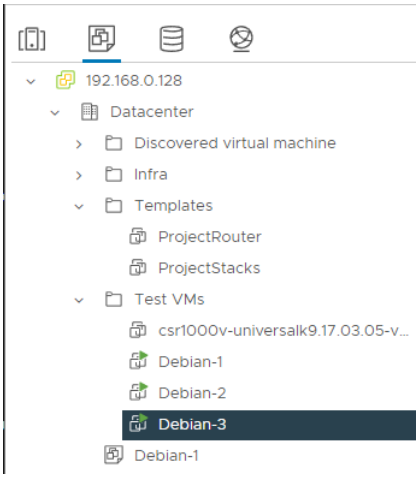

Figure 7.14: vCenter Inventory

Chapter 8

Evaluation

This evaluation will discuss the final solution compared against the original requirements set out in Chapter 4 - Requirements. The project as an ultimate solution will also be reviewed, along with the overall methodologies and technologies that were used to achieve the solution.

8.1 Requirements Evaluation

Both functional and non-functional requirements were set out in Chapter 4 - Requirements. The following sections will discuss how the final solution meets these requirements. A clean installation of the solution was used, with a fresh database so that the application as a whole could be assessed without previous data interfering with the testing process. Tables 8.1 and 8.2 will be used to evaluate the requirements functional and non-functional requirements respectively.

8.1.1 Funtional Requirements

ID	Details	Priority	Met?
FR1	Visual representation of rack space	Must Have	Yes
FR2	Add and remove racks from the space	Must Have	Yes
FR3	Add and remove Terminal Servers from racks	Must Have	Yes
FR4	Add and remove Fabric Nodes from racks	Must Have	Yes
FR5	Add, remove and update projects	Must Have	Yes
FR6	Expand or contract a projects consumption of rack space	Must Have	Yes
FR7	Automate configuration of ACI fabric	Must Have	Yes
FR8	Deploy virtual router using vCenter API	Must Have	Yes
FR9	Deploy virtual services stack to provide remote access VPN	Could Have	No
FR10	Continuous monitoring of ACI and vCenter health	Won't Have	N/A
FR11	Terminal server automated management	Must Have	Yes
FR12	Login system to restrict access	Could Have	No

Table 8.1: Functional Requirements Evaluation

FR1

The solution provides a visual representation of rackspace. Shown in Appendix C Figure C.1.

FR2

The solution allows for the addition and removal of racks from the rackspace. A rack can only be removed if it is not in use by a project. Shown in Appendix C Figures C.2 and C.3.

FR3

The solution allows for the addition and removal of Terminal Servers from racks. A Terminal Server can only be removed if it is not in use by a project. Shown in Appendix C Figure C.4.

FR4

The solution allows for the addition and removal of Fabric Nodes from racks. A Fabric Node can only be removed if it is not in use by a project. Shown in Appendix C Figure C.4.

FR5

The solution allows for the addition, removal and updating of projects. Shown in Appendix C Figures C.5 and C.6.

FR6

The solution allows for the expansion or contraction of a project's consumption of rack space. Shown in Appendix C Figure C.6.

FR7

The solution automates the configuration of the ACI fabric to facilitate communication between the fabric nodes that are members of racks.

FR8

The solution deploys a virtual router using the vCenter API. The configuration of the virtual router is also provisioned from the solution.

FR9

The solution does not deploy a virtual services stack to provide remote access VPN. This is due to the time constraints of the project.

FR10

The solution does not provide continuous monitoring of ACI and vCenter health. This is due to the time constraints of the project and the extra design complexities that would have been incurred.

FR11

The solution automatically provisions terminal servers based on their attached rack. Whilst addition and deletion of terminal servers is possible, due to time constraints, there is no method to update a terminal server. If a terminal server must be updated, then it has to be removed and then re-created. Shown in Appendix C Figure C.7.

FR12

The solution does not include a login system due to time constraints. This would have been a useful feature to have, as it would have allowed for the restriction of access to the solution, however, network access restriction can be used initially in the deployment.

8.1.2 Non-Functional Requirements

ID	Details	Priority	Met?
NFR1	Must be easy to use for staff with less technical knowledge	Must Have	Yes
NFR2	The system status should be easily visible to staff (e.g. errors, project status)	Must Have	Partial
NFR3	The system should be able to easily integrate with existing ACI fabric deployments	Could have	Partial

Table 8.2: Non-Functional Requirements

NFR1

The solution provides an easy-to-use interface, and with some basic explanation as to the principle of operation, most users would be able to use it with ease. At-a-glance metrics are available and the overall utilisation of the rackspace is shown. Whilst knowledge of ACI and vCenter is required to get the solution to a working state, using the system in a day-two scenario will not require this same level of knowledge.

NFR2

The status indicators adjacent to each project show the status of the project throughout the deployment phase of the project, which keeps the user up-to-date with the progress of the automation scripts. The solution does not provide continuous monitoring, so if a problem develops after the deployment phase, then the status displayed will not reflect this.

NFR3

The solution will be able to integrate into existing ACI fabric deployments, however certain fabric functionality like VMware integration must be used. Only single pod deployments are supported, and the solution must use ACI version 5.2(4d) as that is the version that the solution has been developed and tested with.

Overall Requirements Evaluation

Overall, the solution satisfied all of the key functional requirements that were set out in the design of the solution. The main missing features are continuous monitoring of status and a login system. If more time were available, then the features could have been implemented. In the future, a login system can easily be added thanks to Laravel's inbuilt session management system and a suite of libraries and extensions that make it easy to integrate into other login systems such as using LDAP.

The status implementation could also have been improved via the use of WebSockets so that the client doesn't have to constantly poll the server for updates. This would have been a more efficient way of implementing the status system, however, due to time constraints, it was not possible to implement this feature.

8.1.3 Project and Time Management

Kanban was chosen to manage the project as agile would have been too complicated for a single-person development team. Whilst initially useful, usage of the Kanban board drifted due to the extra time required to log and keep track of issues, when they could just be fixed in real-time during development. If the project were to be repeated, then a more concerted effort to make use of Kanban would be made. This is because it would have helped the project's time efficiency to focus on specific features instead of taking a more random approach.

Overall, time was well managed, with most of the literature review being completed before December. Development then continued at a steady pace, with a lot of progress made in March specifically. This is because a lot of the groundwork put in place in the earlier months was able to be connected when the ACI API calls were implemented. The report writing could have been more consistent throughout development, however, a focus on developing the solution was deemed important as unexpected problems and issues could have been encountered.

Chapter 9

Conclusion

The goal of this project was to create an automation platform that aims to streamline the process of managing testbed projects within a lab environment. Through the use of ACI and vCenter to provide infrastructure, the automation platform was able to successfully automate network deployment and virtual machine provisioning. The solution can be attached to a suitably designed ACI and vCenter environment and deploy a project to the desired rackspace with no manual configuration of the infrastructure required. A web interface is presented to the user which provides a simple method to recreate the rackspace virtually inside the application, and then deploy projects to the virtual racks which correspond with the physical racks in the lab. This allows for the rackspace usage of the environment to be easily viewed which will easily help influence how future projects are deployed to maximise space utilisation efficiency.

Whilst the requirement for automated virtual machine deployment was met and the virtual router is deployed automatically, currently the services stack VM is not deployed automatically. This results in manual deployment and IP address configuration of the services VM to get remote VPN connectivity and other services such as DNS running within the testbed.

The primary goal of reducing the amount of time spent when preparing infrastructure for a project was successfully met, with only a minute or so required to enter the required information into the web UI, and a further 5 minutes of deployment time which takes place in the background. This is a significant improvement over the previous method of manually configuring the infrastructure. A need for configuration management has also been successfully eliminated, as ACI and vCenter are now configured automatically and have built-in backup and restoration tools in case any configuration is accidentally modified.

Whilst the solution doesn't continuously monitor the state of the infrastructure, as ACI and vCenter aggregate metrics of the connected devices and are a requirement of the solution,

it would be easier to setup an external monitoring solution such as Observium.

9.1 Future Expansion

As the solution has been built with Laravel and Next.js, it is easy to expand on the current level of functionality in the future. Features such as the login system and continuous monitoring were excluded due to time restrictions but would be first on the priority list if development were to continue. Deployment of a services stack to provide VPN and DNS would also be very beneficial to the time and efficiency of the lab and is also a high priority for future development.

A method to easily deploy the solution via a container orchestration system such as Docker Compose would also be beneficial to add in the future so that one command can be used to deploy the solution to a server. This would also make upgrading the solution easier, as the container image could be updated and then redeployed whilst persisting files stored in the database.

Support for IPv6 would also be beneficial, as adoption is going to increase in the future as the IPv4 space is further depleted. This will also correlate with an increase in project requirements for IPv6 connectivity.

9.2 Learning Points

If this project was to be restarted, several mistakes that were made could be avoided to improve the development experience. Making use of Kanban more effectively, and even tying in with GitHub so that commits can be associated with jobs on the board would be advantageous. This is because project development performance can be easily viewed, and future features can be prioritised and have the correct amount of time allocated. This would also be beneficial if the project were to be worked on by multiple programmers in the future.

Whilst overall time management was good, if development were to be repeated, then a more concerted effort to distribute development throughout the project timeline would have been beneficial. This would have allowed possibly all of the features to have been developed and would have allowed more time for testing and bug fixing.

References

Alizadeh, M., & Edsall, T. (2013). On the data path performance of leaf-spine datacenter fabrics. *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, 71–74. https://doi.org/10.1109/HOTI.2013.23

Alsaeedi, M., Mohamad, M. M., & Al-Roubaiey, A. A. (2019). Toward adaptive and scalable openflow-sdn flow control: A survey. *IEEE Access*, *7*, 107346–107379. https://doi.org/10.1109/ACCESS.2019.2932422

Antón, A. (1997). Goal identification and refinement in the specification of information systems. *PhD Thesis, Georgia Institute of Technology*.

Atlassian. (2023). *Trello brings all your tasks, teammates, and tools together*. Retrieved January 3, 2023, from https://trello.com/

Bhardwaj, R. (2020). Opflex. https://ipwithease.com/opflex/

Borgenstrand, M. (2018). Network automation – the power of ansible (dissertation). http://urn.kb.se/resolve?urn=urn:nbn:se:miun:diva-34002

CDW. (2015). The future of networking arrives. *Commun. ACM*, 16–19. https://webobjects.cdw.com/webobjects/media/pdf/CDWCA/White-Paper-The-Future-of-Networking-Arrives-MKT2862CA.pdf.

Collin, W. (2021). Automation in multi-domain software-defined networking: Overview and use cases.

Duffy, J. (2014). Cisco reveals openflow sdn killer; opflex protocol for aci offered to ietf, opendaylight. *Network World*.

Ieee standard glossary of software engineering terminology. (1990). *IEEE Std 610.12-1990*, 1–84. https://doi.org/10.1109/IEEESTD.1990.101064

Ijari, P. (2017). Comparison between cisco aci and vmware nsx. *IOSR Journal of Computer Engineering (IOSR-JCE)*, *19*(1), 70–72. https://www.researchgate.net/profile/Palash-Ijari/publication/314082881_Comparison_between_Cisco_ACI_and_VMWARE_NSX/links/5c127c74299bf139c756b2dc/Comparison-between-Cisco-ACI-and-VMWARE-NSX.pdf.

Kirkpatrick, K. (2013). Software-defined networking. *Commun. ACM*, *56*(9), 16–19. https://doi.org/10.1145/2500468.2500473

Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Latif, Z., Sharif, K., Li, F., Karim, M. M., Biswas, S., & Wang, Y. (2020). A comprehensive survey of interface protocols for software defined networks. *Journal of Network and Computer Applications*, *156*, 102563. https://doi.org/https://doi.org/10.1016/j.jnca.2020.102563

Networking, L. F. (2021). Odim introduction. https : / / wiki . lfnetworking . org / download / attachments / 50528912 / ODIM % 20Introduction . pdf ? version = 1 & modificationDate=1612306983000&api=v2

Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, *16*(3), 1617–1634. https://doi.org/10.1109/SURV.2014.012214.00180

Pham, M., & Hoang, D. B. (2016). Sdn applications - the intent-based northbound interface realisation for extended applications. *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 372–377. https://doi.org/10.1109/NETSOFT.2016.7502469

Project, O. (2023). Resource aggregator for odim. https://github.com/ODIM-Project/ODIM

Rana, D. S., Dhondiyal, S. A., & Chamoli, S. K. (2019). Software defined networking (sdn) challenges, issues and solution. *International journal of computer sciences and engineering*, *7*(1), 884–889.

Sabharwal, N., & Wadhwa, M. (2014). *Automation through chef opscode: A hands-on approach to chef*. Apress. https://books.google.co.uk/books?id=umwLBAAAQBAJ

Seyedebrahimi, M., Raschellà, A., Bouhafs, F., Mackay, M., Shi, Q., & Eiza, M. H. (2016). A centralised wi-fi management framework for d2d communications in dense wi-fi networks. *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, 1–6. https://doi.org/10.1109/CSCN.2016.7785187

Sokappadu, B., Hardin, A., Mungur, A., & Armoogum, S. (2019). Software defined networks: Issues and challenges. *2019 Conference on Next Generation Computing Applications (NextComp)*, 1–5. https://doi.org/10.1109/NEXTCOMP.2019.8883558

teamgantt. (2023). *Teamgantt is the refreshing solution that brings project scheduling software online*. Retrieved January 3, 2023, from https://trello.com/

Terraform. (2023). Cisco aci provider. https : / / registry . terraform . io / providers / CiscoDevNet/aci/latest/docs

Vasconcelos, C. R., Gomes, R. C. M., Costa, A. F. B. F., & da Silva, D. D. C. (2017). Enabling high-level network programming: A northbound api for software-defined

networks. *2017 International Conference on Information Networking (ICOIN)*, 662–667. https://doi.org/10.1109/ICOIN.2017.7899569

Wågbrant S, D. R. V. (2022). Automated network configuration: A comparison between ansible, puppet, and saltstack for network configuration. *Dissertation*. http://urn.kb.se/resolve?urn=urn:nbn:se:mdh:diva-58886.

Wazirali, R., Ahmad, R., & Alhiyari, S. (2021). Sdn-openflow topology discovery: An overview of performance issues. *Applied Sciences*, *11*(15). https://doi.org/10.3390/app11156999

Xu, J., & Russello, G. (2022). Automated security-focused network configuration management: State of the art, challenges, and future directions. *2022 9th International Conference on Dependable Systems and Their Applications (DSA)*, 409–420. https://doi.org/10.1109/DSA56465.2022.00061

Yan, Q., Yu, F. R., Gong, Q., & Li, J. (2016). Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys and Tutorials*, *18*(1), 602–622. https://doi.org/10.1109/COMST.2015.2487361

Zhou, W., Li, L., Luo, M., & Chou, W. (2014). Rest api design patterns for sdn northbound api. *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 358–365. https://doi.org/10.1109/WAINA.2014.153

Appendix A

Project Initiation Document

School of Computing
Final Year Engineering Project

Project Initiation Document

Matthew Gaynor

A Web Driven SDN Orchestrator For The Provisioning of ACI Fabric and Lab Infrastructure

1. Basic details

Student name:	Matthew Gaynor
Draft project title:	A Web Driven SDN Orchestrator For The Provisioning of ACI Fabric and Lab Infrastructure
Course and year:	Computer Networks BSc 2023
Project supervisor:	Dr Shikun Zhou
Client organisation:	Cisco - CX Labs UK
Client contact name:	David Smith

2. Degree suitability

This project will aim to solve a problem that has arisen due to a lack of automation and an ever-aging infrastructure that is failing to keep up with the times.

The project will be centred around Cisco ACI, which is Cisco's datacentre software defined networking (SDN) solution. ACI manages the network devices which interconnect to form a fabric, that can be managed from one web UI and via a REST API. This ties together both software and networking to form one solution that is highly scalable and ready for the current need to be application centric.

ACI utilises VXLAN and IS-IS under the hood, so it is required to have an understanding as to how this works for troubleshooting.

There will also be a heavy use of APIs, with a custom backend being required to interact and store data from ACI, as well as vCenter for virtualisation automation.

3. Outline of the project environment and problem to be solved

The client is CX Labs within Cisco Systems. Due to the nature of Cisco being a large multi-national company, the company is split up into different business units. These business units specialise in different fields and provide different services. CX Labs provides lab space mainly to internal customers, including over divisions of CX (Customer Experience) Common use of lab space includes testing and validation services, proof of concepts and replications of client's networks for testing. In some cases, customers can host equipment directly in the labs and have remote access to the stored equipment via the means of a VPN connection to the lab.

Over the years, CX Labs has conventionally had more isolated lab space compared to DMZ space. Whilst the isolated environment is good, it is only accessible to employees of Cisco. Because of this limitation, demand for DMZ space has increased, meaning lab space that has connectivity to the internet. Customers increasingly want access to the testbeds they are paying for to carry out their own testing. The nature of this DMZ space means physical separation from the rest of the network to prevent accidental bridges between un-protected internet and the corporate network. Due to the increasing demand, more lab space is being converted to DMZ space, this has resulted in the need for new infrastructure to support the additional space.

The existing DMZ lab space uses an old approach using Nexus 9K switches at the core, and old Catalyst 2960 switches for Top of Rack. Whilst solidly reliable, to deploy a new project, it is a very involved process, with VLANs, trunks, access ports, vCenter Port Group creation, routing and VPN all having to be configured and provisioned manually.
This environment commonly results in configuration mistakes which can result in extra time required for troubleshooting, and in some cases, inter-project communication which should not be possible. It is also very time consuming which stretches the limited time availability when onboarding new projects to the limit.
Another issue that arises when this creates is that when a project is terminated, the same work is then required in reverse to free infrastructure up for the next project in the lab. Again, this resulting in configuration errors, and in some cases, the project infrastructure is never properly decommissioned due to time constraints. This leads to more unexpected work when a new project comes in and requires the lab space.

4. Project aim and objectives

The aim of the project is to deliver a web dashboard that will allow the lab team to deploy projects into the lab space with minimal involvement in configuring the networking equipment. The dashboard will be powered by NextJS for the frontend and Laravel for the backend. The Laravel backend will interact with the vCenter and ACI APIs to provide the network and service automation. Laravel will also store the state of the lab and keep track of IP addressing and IDs to improve ease of use and response times. It will serve as an overall orchestration master for the lab.

- Web UI for management of rack space and infrastructure
- Automated routing deployment using the CSR1000v virtual router platform
- Automated EPG, BD and VRF deployment using ACI.
    - ToR port allocation also automated
- Terminal Server automation to place terminal server in correct network
- Ability to manage the space from the Web UI, including the provisioning of additional racks, Terminal Servers and ToR switches.

5. Project constraints

Equipment is being loaned and hosted by Cisco, could be affected by organisation changes and restructuring. Members of staff leaving could also impact the project.
Not able to test at scale, only able to simulate 3 racks, with a collapsed spine and leaf architecture.

6. Facilities and resources

All resources required for this project are provided by myself and Cisco. I have a testbed of equipment that will be necessary to complete the project with dedicated external IP addresses and VPN access.

7. Log of risks

Description	Impact	Mitigation/Avoidance
Moving equipment around may result in injury	Cause delays to the project timeline	Follow the appropriate safety and manual handling guidance when lifting equipment.
Changing in Cisco policy may mean I am no longer able to access lab	Cause delays to the project and result in the inability to complete testing	Take backups of virtual machines and keep in contact with managers at Cisco. Worst case, all of lab can be simulated and done virtually.
Hardware Failure	Inability to continue testing using the testbed	Take config backups and arrange for replacement devices to be installed or move to virtualised devices. Use Git for version control to keep codebase backed up to GitHub.

8. Project deliverables

The result will be a web UI with a backend that handles the interactions with the networking equipment. The web UI will be easy to use and have a good UX design to allow for a minimal learning curve and be easy for someone with limited networking knowledge to use.
The backend will take care of automation and the only manual input from the user will be the IP addresses of the relevant equipment, such as the APIC and terminal servers.

Documentation including a proposed network topology and user guide will be included. As well as basic installation instructions, although Docker will be used to simplify this as much as possible.

9. Project approach

As this will be a software development project, I will be using Trello to manage my development. Trello is a web application that provides a way to manage a Kanban SDLC. As it is just me developing the software, there is no need for a full agile methodology to be implemented and doing so would result in wasted time, so Kanban is the perfect choice.
I will also follow the waterfall model as a guide to the overall development cycle.

As I will be utilising the vCenter and ACI REST APIs, the appropriate documentation for these must be consulted in order to implement them. Elements such as schema and authentication

will vary from API to API, so the documentation is required for this. Both VMware and Cisco provide detailed documentation on their APIs.

As I am building my own REST API, it will be best practise to follow the OpenAPI specification. Whilst there is no one specific REST standard, the OpenAPI standard is widely regarded as the most optimal when creating REST APIs.

10. Project plan

The first step is to carry out a literature review to determine existing solutions and work out their advantages and shortcomings. This will allow me to put together some best practises to follow and allow me to make the most optimal solution.

The next step will then be performing a detailed requirement analysis to determine the exact requirements of the project. I will then be able to create a success criteria which will outline the key functionality that is required from the project.

The next phase will then be taking the success criteria and designing how the artefact will work from a technical standpoint. This will include designing the UI/UX and how the database will store information. An API design will also help determine the models required to handle the data. I will also need to research the ACI and vCenter APIs which I have used briefly in the past, but more in-depth knowledge will be required for this.

After the design phase is complete, the artefact can be created which will involve writing all the code and documenting the process.

The application can then be tested by the end users and by also using traffic generation to ensure that the automation aspect has carried out its intended functions.

11. Supervisor Meetings

We have agreed that a weekly sync-up would be a good frequency for us. Supervisor input shouldn’t be needed too frequently, so periods of leave won’t be a problem and I can plan upcoming questions accordingly. Meetings will take place face-to-face.

12. Legal, ethical, professional, social issues (mandatory)

As the artefact will be automating the network infrastructure that will be supporting sensitive testbeds, it is important the security of the platform is taken deeply into consideration. As such, authentication will be required against authentication servers to ensure that only authorised users have access to the dashboard. Testing of the provisioned network policy must also be tested to ensure that no cross-communication between projects occurs. This is very important as ARP leaking and communication could result in sporadic issues and hard to troubleshoot problems. However, as the environment is not production and is purely for testing, security is not as crucial as a production piece of software would required to be.

As the platform will be automating network device configuration, it will remove the need for as much time to be spent by an engineer, which has the potential to devalue their skill sets and appear to higher management like their role is no longer required.

Appendix B

Ethics Review Certificate

Certificate of Ethics Review

Project title: A Web Driven SDN Orchestrator For The Provisioning of ACI Fabric and Lab Infrastructure

Name:	Matt Gaynor	**User ID**:	922830	**Application date**:	10/10/2022 11:18:58	**ER Number**:	TETHIC-2022-103684

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Haythem Nakkas, Dalin Zhou**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Dr Shikun Zhou**
Is the study likely to involve human subjects (observation) or participants?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

Appendix C

Web Interface Screenshots

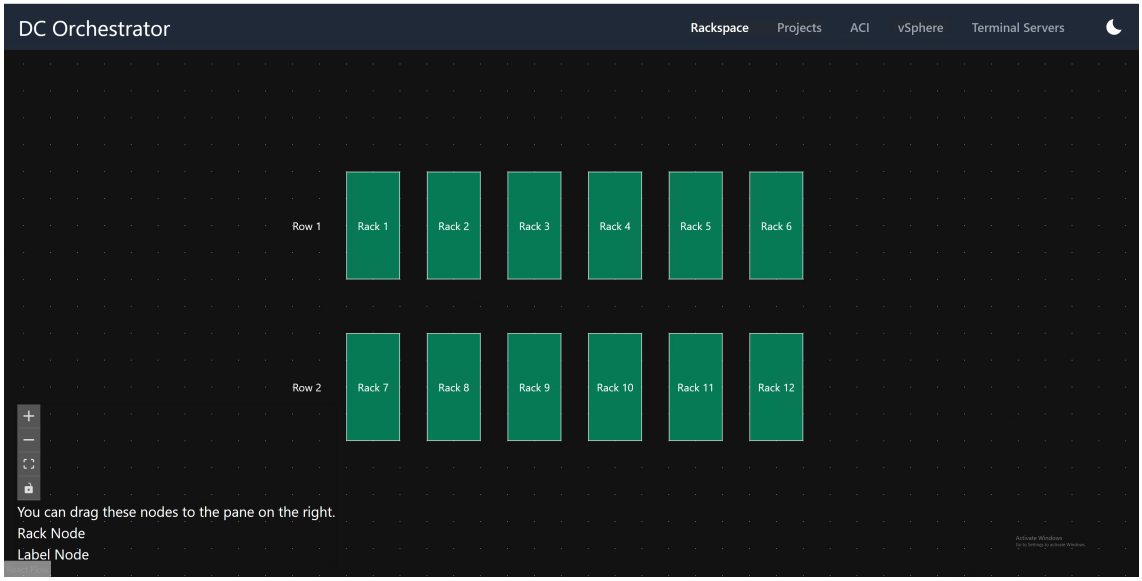

Figure C.1: Rackspace View

Figure C.2: Rackspace Edit and Delete

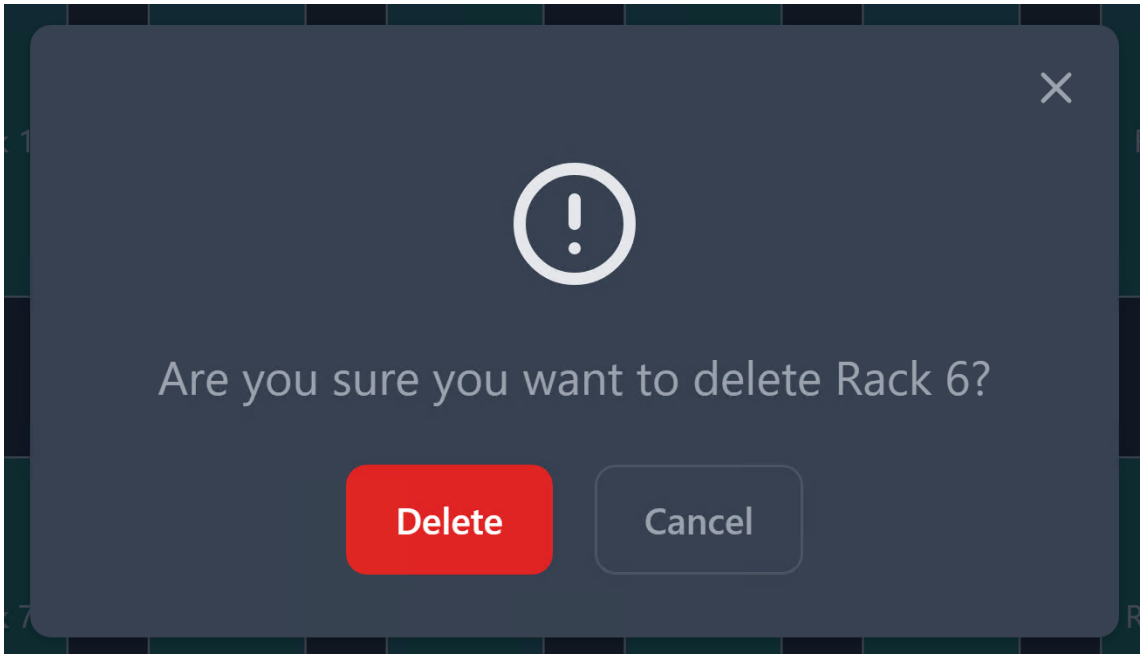

Figure C.3: Rackspace Delete

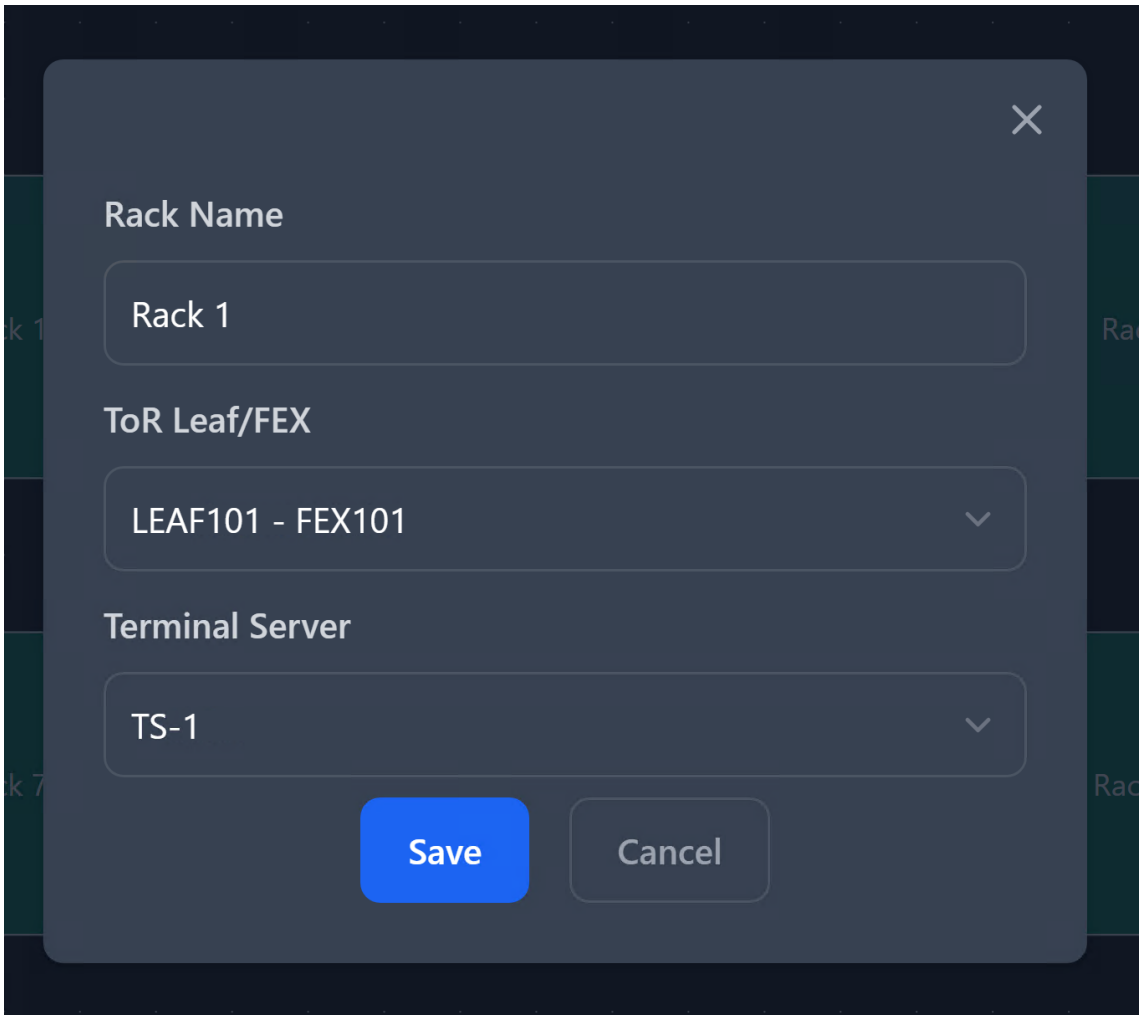

Figure C.4: Rackspace Edit

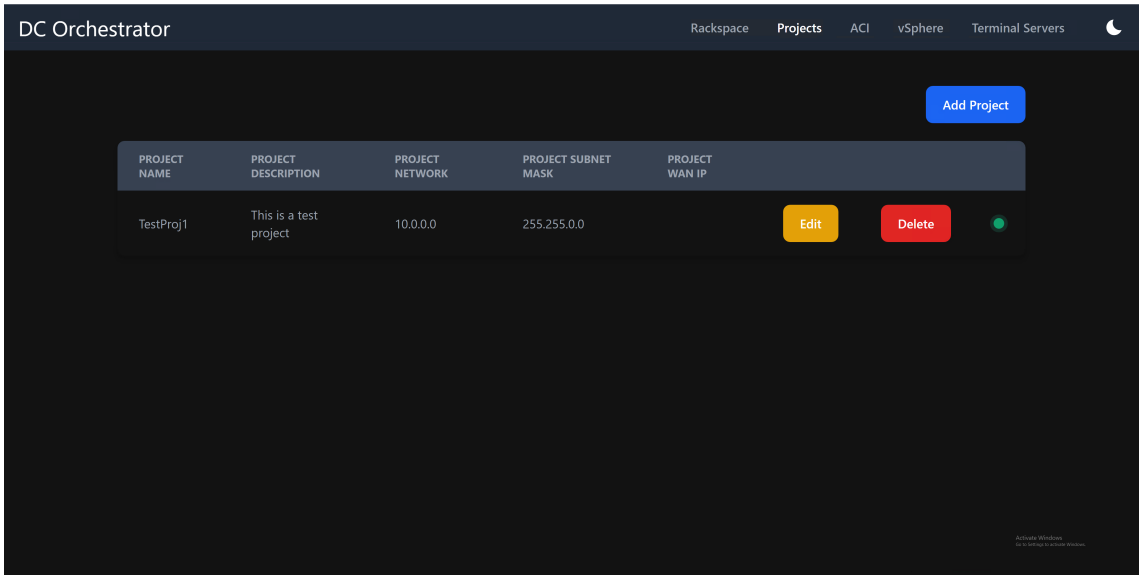

Figure C.5: Project View

Figure C.6: Project Edit

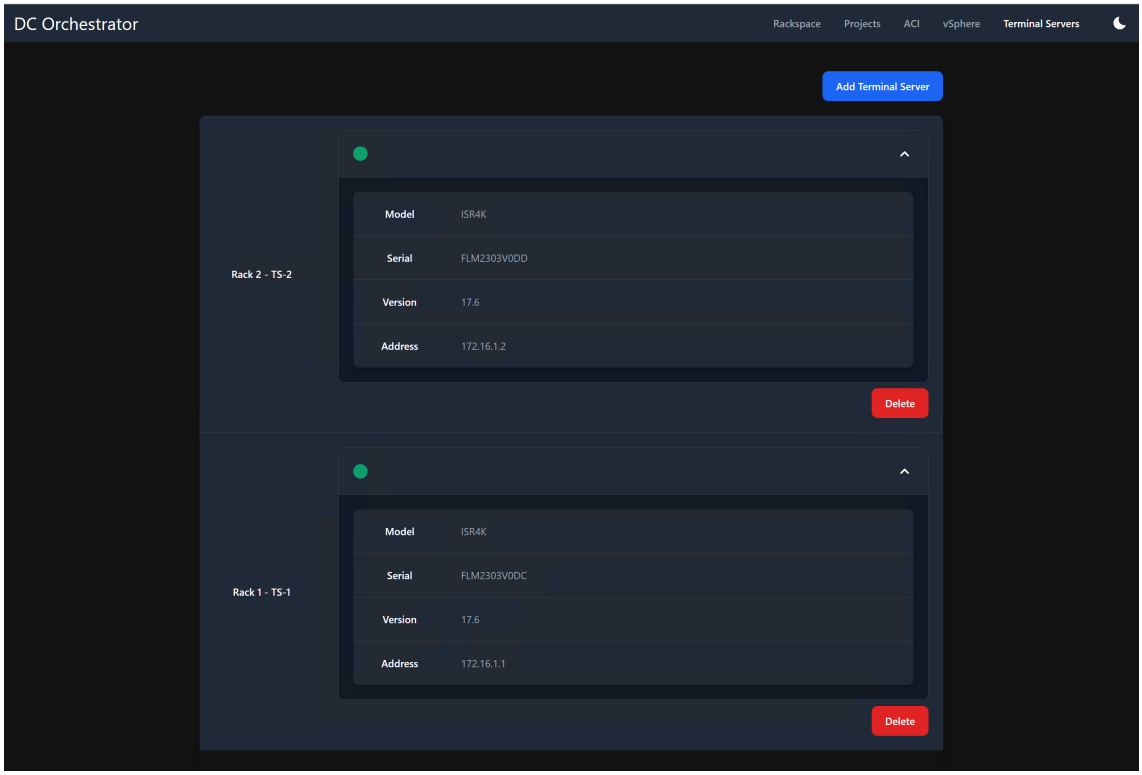

Figure C.7: Terminal Server Management

Appendix D

Installation Guide

This guide will outline the requirements and installation procedure to get the automation platform up and running. The guide will assume that the user has a basic understanding of Linux and the command line as well as Cisco ACI and VMware vCenter.

This solution is designed to allow a testing environment to provide OOB connectivity to a variety of projects within a testing environment through the use of a web UI. It achieves this through the use of Cisco ACI and VMware vCenter. Each rack within the lab space must have its own dedicated FEX or leaf switch as well as a terminal server, although a rack can exist without either. When racks are selected to be part of a project, the automation platform will deploy L2 connectivity to any FEX and leaf that belong to the selected racks, as well as including the terminal servers into this L2 domain. A virtual router will then be deployed on vCenter which will have the same aforementioned L2 connectivity to the selected FEXs and leafs, as well as connectivity to the desired WAN uplink.

Requirements

- *Cisco ACI v5.2(4d)*
- *VMware vCenter 7.0.3*
- *ESXi 7.0.3 Host (at least one)*
- *CSR1000v 17.03.05*
- *Terminal Servers (Must be IOS-XE 17.06.03a)*

The solution will require some existing ACI configuration to be in place. The following will be required:

- *VMM integration between ACI and vCenter*

- *EPGs for terminal servers, internet connectivity, virtual routers and management VMs.*
  - *The virtual router EPG must have a DHCP server to automatically assign IP addresses to the virtual routers.*
- *An EPG that has access to the virtual router and terminal server EPGs, access to ACI and vCenter APIs can either be in-band or out-of-band.*
- *A static VLAN pool to be used by the automation platform, this should have a unique set of VLANs from any other VLAN pool to prevent issues from arising.*
- *An interface selection policy per leaf/FEX should be created, as this will need to be associated with each node via the automation platform web UI.*

An example design of both the physical deployment and the ACI configuration can be seen in Figure D.1 and Figure D.2 respectively.

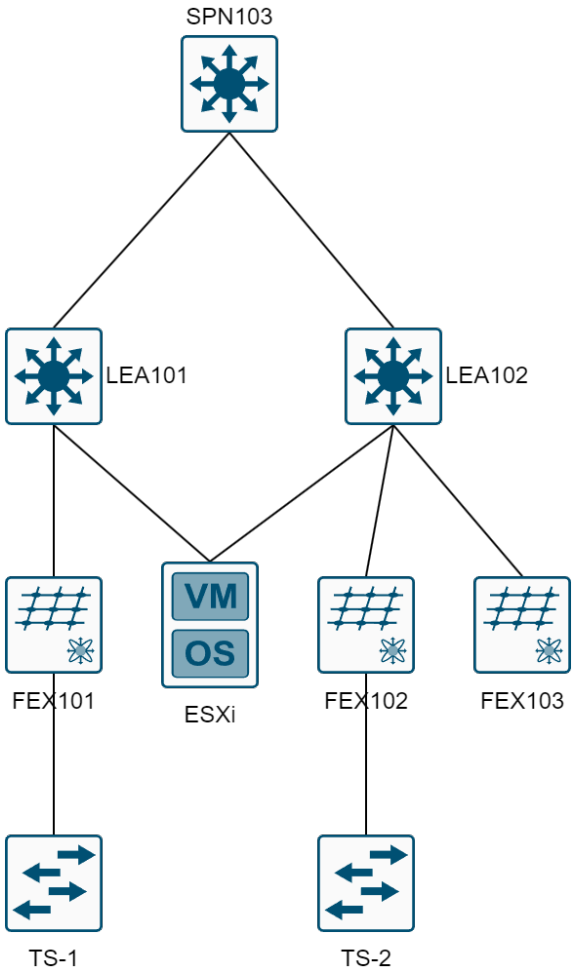

Figure D.1: Example physical deployment

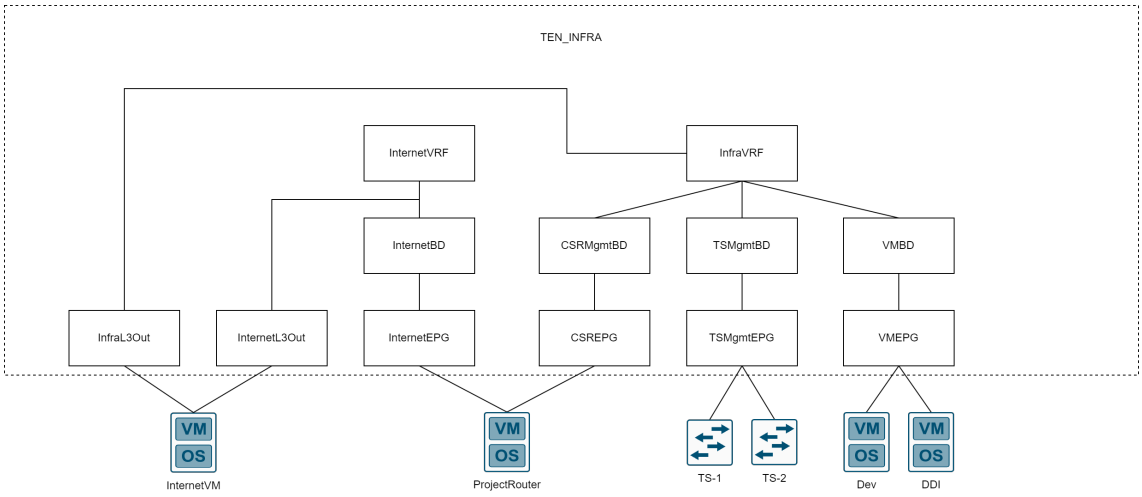

Figure D.2: Example ACI configuration

ENV File Configuration

For the automation platform to operate correctly, the .env file must be configured with the required information so that various services such as ACI and vCenter can be accessed correctly. A breakdown of the required .env file variables is provided below:

Variable	Example	Description
APIC_IPADDR	192.168.0.125	IP address of the APIC controller
APIC_USERNAME	admin	Username for the APIC controller
APIC_PASSWORD	password	Password for the APIC controller
ACI_POD	1	The pod number of the ACI fabric that the automation platform will automate
ACI_VMWARE_DOMAIN	ACI-DVS	The name of the VMM integration domain
ACI_INFRA_DOMAIN	InfraPhys	The name of the physical domain used to connect terminal servers to the ACI fabric
ENHANCED_LACP	LACP	Name of the enhanced LACP policy used to connect ESXi nodes, leave this null if Enhanced LACP is not being utilised
VSPHERE_IPADDR	192.168.0.128	IP address of the vCenter server
VSPHERE_USERNAME	admin	Username for the vCenter server
VSPHERE_PASSWORD	password	Password for the vCenter server
PROJECT_ROUTER	ProjectRouter	Name of the virtual router VM template
PROJECT_ROUTER_USERNAME	automation	Username that will be used to connect to virtual router VM
PROJECT_ROUTER_PASSWORD	password	Password that will be used to connect to virtual router VM

Virtual Router Template

The virtual router template should be a powered off VM not a template, this is due to a limitation in the vCenter REST API. The VM should have the following interface assignments:

Interface	Port Group
Network Adapter 1	quarantine
Network Adapter 2	Internet EPG
Network Adapter 3	Virtual Router Management EPG (with DHCP)

Network adapter 1 will automatically be assigned to the project EPG by the automation scripts. The automation script will automatically configure NAT, default route and WAN

IP address, so only the 3rd network adapter needs to be configured to retrieve its IP address from DHCP. RESTCONF will also need to be enabled, sample configuration is shown below.

```
vrf definition Mgmt
 address-family ipv4
 exit-address-family
!
interface GigabitEthernet3
 vrf forwarding Mgmt
 ip address dhcp
 negotiation auto
 no mop enabled
 no mop sysid
!
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
ip http secure-port 1025
restconf
```

Additional ACLs should be configured to secure access to RESTCONF, however, these will depend on the specific environment and deployment. Once the initial configuration is complete, the VM should be powered off and left in the desired folder where the project routers will be stored.

Solution Deployment

In the future, the solution will be packaged via Docker Compose, which will allow one command to be run to install and serve the solution as a whole. With ACI and vCenter configured, the automation platform can be deployed. The platform will need Docker running ideally ontop of a Linux host, however, it can be deployed on Windows using Docker Desktop. NodeJS and NPM will also be required to build and serve the frontend. The following steps will need to be completed to deploy the platform:

1. *Clone the repository from GitHub*
2. *Configure the .env file with the required information*
3. *Run the following command in the backend folder to obtain the required composer dependencies:*

```
docker run --rm \
    -u "$(id -u):$(id -g)" \
    -v "$(pwd):/var/www/html" \
    -w /var/www/html \
    laravelsail/php82-composer:latest \
    composer install --ignore-platform-reqs
```

4. *Run the following command in the backend folder to start the platform:*

```
./vendor/bin/sail up -d
./vendor/bin/sail artisan key:generate
./vendor/bin/sail artisan migrate
./vendor/bin/sail artisan queue:listen --timeout 400
```

5. *Run the following command in the frontend folder to obtain the required node dependencies:*

```
npm install
npm run build
npm run start
```

When first connecting to the solution, navigate to the ACI page and select the VLAN pool that was created in ACI for use by the automation platform. Then click Set Interface Profiles, and assign the corresponding interface profile to each node discovered by the automation platform. Once this is complete, the solution is ready to be used. Figure D.3 shows an example mapping:

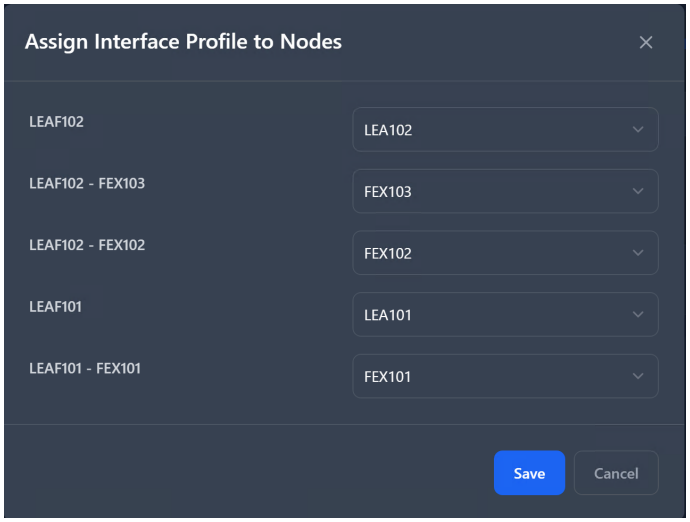

Figure D.3: Example interface profile assignment

Terminal servers should also be added via the terminal servers page, where all information the platform needs is required on the addition form.

Appendix E

User Guide

This guide will cover how to use the solution from a day two perspective after the solution has been integrated with the ACI fabric and vCenter environment. The rackspace page is the main page of the solution, this page will show the rackspace that is currently available and in use by projects. Additional racks can be deployed by dragging a rack node from the bottom left of the screen. A label can also be dragged onto the floorplan to store text that may be deemed useful. A rack can then be assigned a top-of-rack node (either a leaf or a FEX) and a terminal server, this should represent what hardware is physically deployed in the rack to ensure the correct connectivity is deployed to the racks.

A project can then be deployed via the projects page, where all information required by the platform is requested via the form. The racks that the project should occupy can be selected via shift-clicking on the racks to select multiple. In the infrastructure tab, the Project Subnet section is the network that will be used internally by the testbed network, if left blank, then the automation platform will automatically assign a non-overlapping subnet to the project. If a specific subnet is required, then this can be specified. The virtual router will be provisioned to the last available IP address in the subnet, and the terminal servers will be assigned sequential addresses starting from the first available IP address in the subnet. The WAN address for the project is also provided, these addresses will be automatically pushed to the virtual router to provide internet connectivity to the project. This form is shown for reference below.

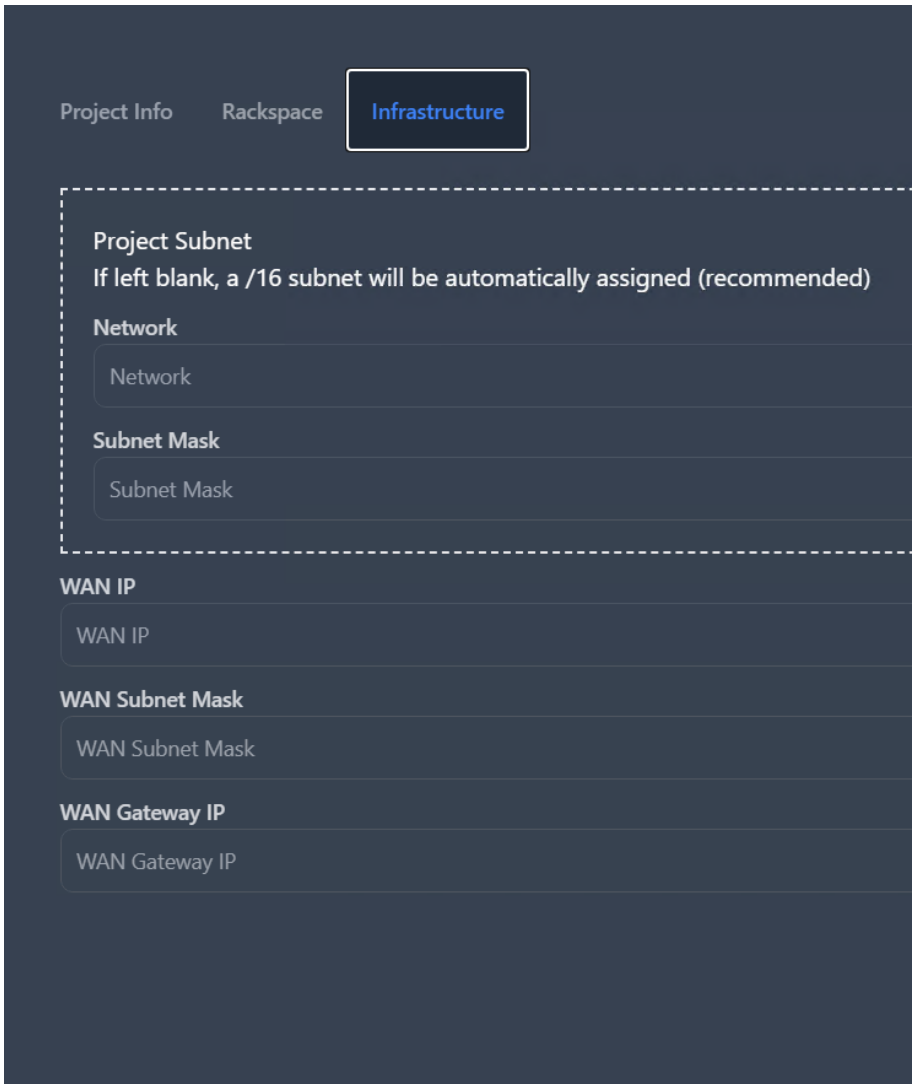

Figure E.1: Project Creation Form - Infrastructure

When the project is deployed, the status of the project will update as the deployment progresses through the ACI and VMware deployment phases. Once deployed, a project can then be edited, to modify its occupation of the rackspace. Racks can either be added or removed to a project, allowing its consumption of rackspace to change throughout the lifecycle of the project. Some aspects, such as the name and IP address cannot be changed, and a project must be deleted and recreated if these need to be changed.