UNIVERSITY OF PORTSMOUTH

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

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)
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarised Zone
DNS	Domain Name System
DPG	Distrubited Port Group
DVS	Distributed Virtual Switch
EPG	Endpoint Group
ERD	Entity Relationship Diagram
FEX	Fabric Extender
HTTP	HyperText Transfer Protocol
LACP	Link Aggregation Control 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

SPA	Singple Page Application
SSH	Secure Shell
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
WAN	Wide Area Network

Abstract

The purpose of this project is to provide a solution to CX Labs UK within Cisco Systems LTD that will streamline the operation of DMZ lab-space. The application shall allow lab staff to prepare network infrastructure for an incoming project with little to no CLI interaction and it shall be web-based. ACI, VMware ESXi and vCenter shall be used to manage the networking and virtual machines respectively. The lab space will be represented by a small testbed containing the minimum quantity of networking equipment and compute resources required to accurately simulate the environment.

Acknowledgements

I would like to thank Cisco, my manager and good friend, Dave Smith, for his close help in allowing me to use equipment necessary for testing and developing the solution.

Consent to Share

I consent / do not consent for this project to be archived by the University Library and potentially used as an example project for future students.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1   The Client

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 provide bespoke testing services to customers wishing to use Cisco's expertise to test a range of situations, from regression and firmware testing to full upgrade and migration plans.

In order to match the customers environment as close as possible, a scaled down version of the customers 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 in order to access testbeds. More and more customers however are requesting remote access to their testbeds. 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.

1.2   The Problem

Currently the solution for the DMZ network infrastructure consists of 4 Nexus 9K devices, with FEXs for RJ45 connectivity and 2960S ToR switches. Whilst this solution is functional and works, the major downside is that there is no automation or configuration management solution deployed. This means over time, configuration drift occurs as manual changes are made, but not made the same to all switches. Unused VLANs are also not removed from the switches which leads to bloat and a larger configuration than is required. The same situation also occurs on vCenter with the configuration of the DPGs within the DVS where unused DPGs are never removed or are labelled incorrectly.

The manual configuration of the required routing and VPN termination as well as other lab requirements such as a NTP, DNS and RADIUS all take a lot of time to configure. This

time could be better utilised, such as preparing the physical rack space for the racking and stacking of new equipment.

1.3   Aims and Objectives

The aim of this project is to provide an easy to use dashboard that allows CX Labs to easily onboard and manage projects. This will be achieved by providing a web based interface that allows the user to create a new project, which will then automatically create the required configuration within ACI, vCenter, and other associated network infrastructure. Management of existing projects will also be supported, operations such as expanding a projects rackspace utilisation, or removing a project from the environment will be supported. The dashboard will also provide a view of the current projects utilisation of the lab space, as well as the current utilisation of the lab space as a whole.

1.3.1   Deliverables

- A web based dashboard that allows the user to manage and provision projects

- User guides

  - User guide for the dashboard

  - User guide for how to provision ACI, vCenter and other associated network infrastructure so that it will be compatible with the automation solution

- A report detailing the design and implementation of the solution

1.4   Constraints

Due to the fact that the testbed utilised for testing and development is in a remote datacenter, physical access to the testbed will be limited. This may result in delays if a problem with the physical infrastructure occurs which could lead to the project running over time. The solution will be designed to be as resilient as possible, but there is a chance that the solution may not be able to recover from a failure. There may also be a chance that access to the testbed is revoked, which would result in the project being delayed until access is restored.

External factors such as the availability of the testbeds internet uplink and power supply may also have an impact on the projects delivery within the required timescale.

1.5   Evaluation

To ensure that the project delivers on the required functionality, the following evaluation criteria will be used to evaluate the project:

- Does the dashboard allow for the easy creation, management and deletion of lab projects?

- Does the dashboard allow for a logical view of the entire lab space as a whole?

- Does the dashboard show project status and health of the associated devices?

- Does the automation platform create a schema in ACI that is easy to follow and troubleshoot should a problem arise?

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, it will also involve analysing best practises 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 for this review will be from relevant books, online websites, professional publications and Request for Comments. Multiple services will be used to identify and obtain sources that can be used for the purposes of this review.

- Google Scholar

- IEEE Explore

- ResearchGate

- University of Portsmouth Library

Research was conducted that was related to the following set of topics:

- What is software defined networking?

- What types of software defined networking exist?

- What are the advantages and disadvantages?

2.1   Definition of Software Defined Networks

All 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".
SDN is often defined using four pillars, Kreutz et al. (2015) define these as the following:

1. "The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements"

2. "Forwarding decisions are flow based, instead of destination based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions)"

3. "Control logic is moved to an external entity, theso-called SDN controller"

4. "The network is programmable through software applications running on top of the NOS that in-teracts with the underlying data plane devices."

2.2 Overview of Software Defined Networks

Since this project is centred around software defined networking and the automation of these networks, it is critical to ensure that the principles and their method of operations are understood.

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. (2014) go 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". This leads to one of the solutions, Software Defined Networking. "Software Defined Networking (SDN) is a new networking paradigm which the forwarding hardware is decoupled from control decisions. It promises to dramatically simplify network management and enable innovation and evolution". Software defined networking "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

2.3 Types of Software Defined Networks

Software Defined Networking is a blanket term for any solution that tackles conventional networking with a programmatic view. There are "the two main delivery models: Imperative and declarative" CDW (2015). CDW (2015) go on to state that Imperative SDN is where "A centralized controller (typically a clustered set of controllers) functions as the network's 'brain'" and that declarative SDN is where "the intelligence is distributed out to the network fabric. While policy is centralized, policy enforcement isn't". CDW (2015) give "a protocol such as OpenFlow explicitly telling network switches precisely what to do and how to do it" to be an example of imperative SDN.

Chapter 3

Methodology

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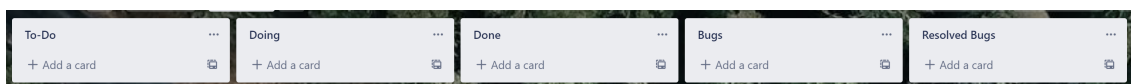
Figure 3.1: Trello Kanban board used to manage the development process

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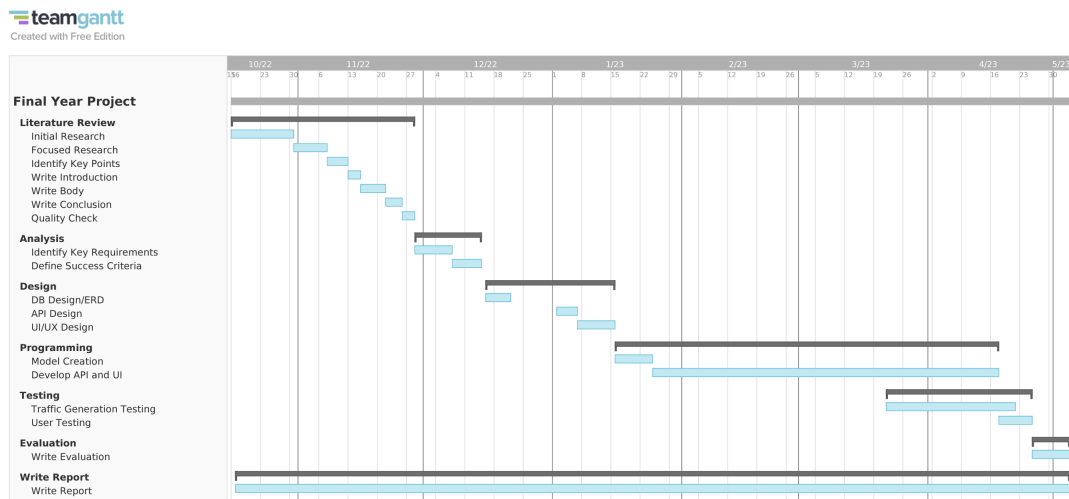
Figure 3.2: Gantt chart used to outline the key project milestones

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 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 in more detail 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. The 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 a 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. 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 in the application, 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 are required in the rack space.

FR5 provides the ability to manage projects present in the rack space and will provide the core automation functionality.

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 nonbehavrioral 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. By abstracting away the networking and configuration through the use of automation, only a basic understanding of networking should be required for the platform to be used.

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 network configuration that is required to automate the deployment of projects to the network.
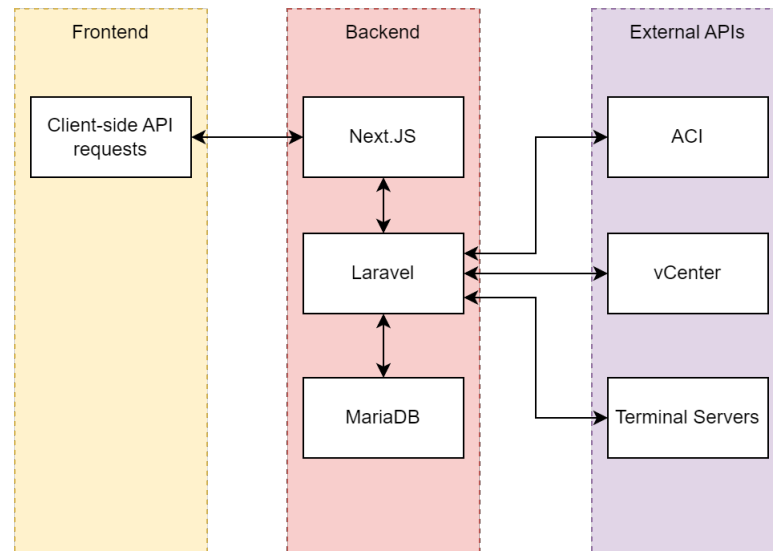
Figure 5.1: Web Architecture Design

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.

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.

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.
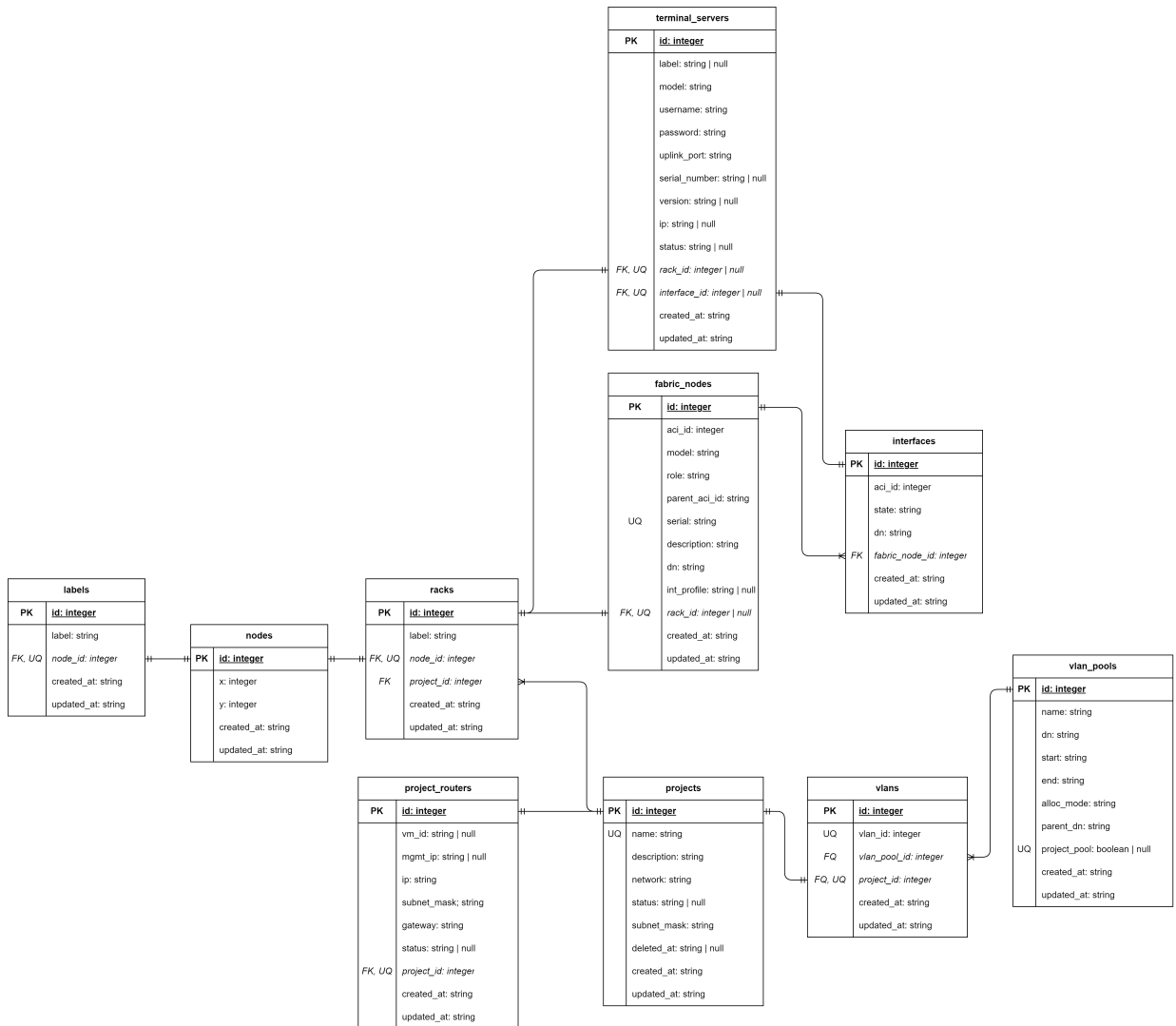

Figure 5.2: Database ERD

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 side. 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 get 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
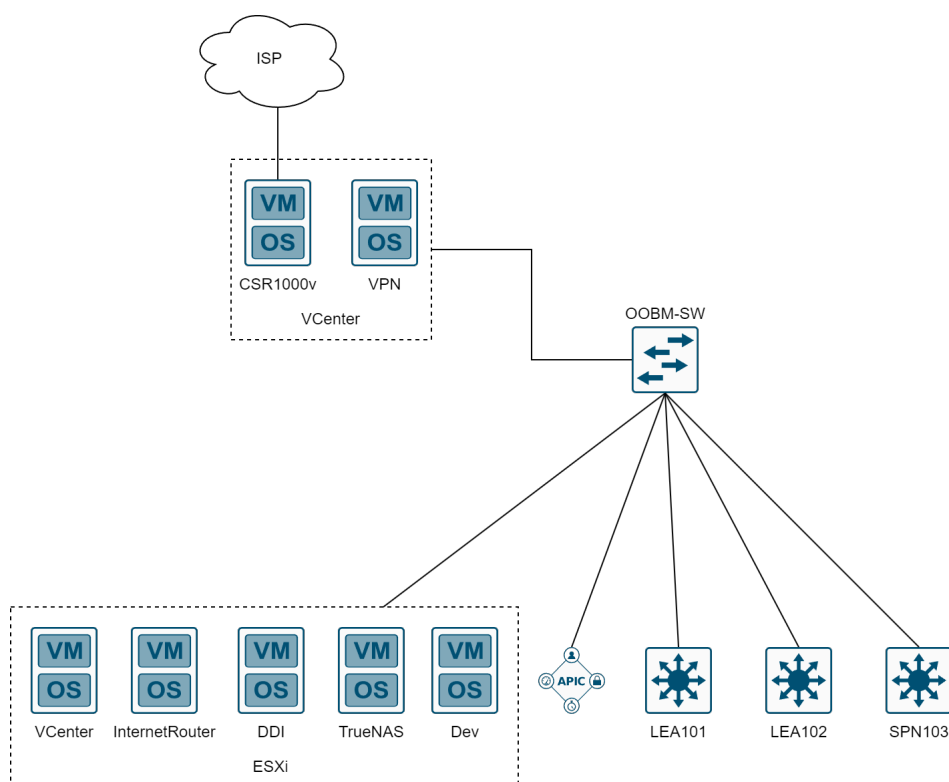ESXi server are also detailed in the topology shown in figure 5.3

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.
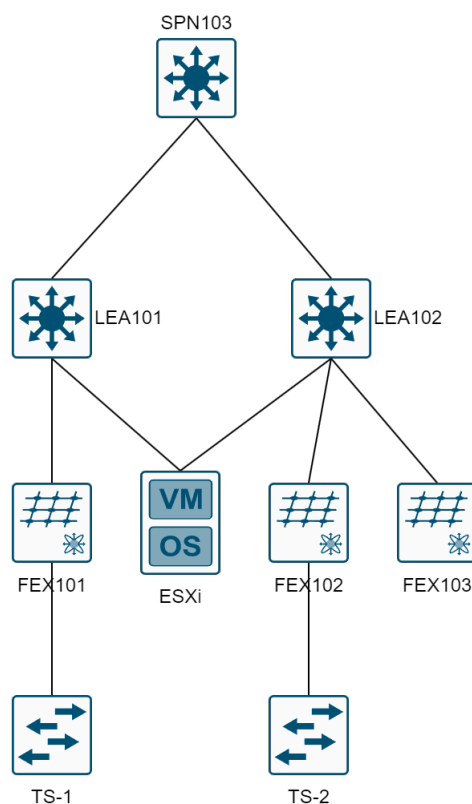

Figure 5.4: Fabric Topology

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.
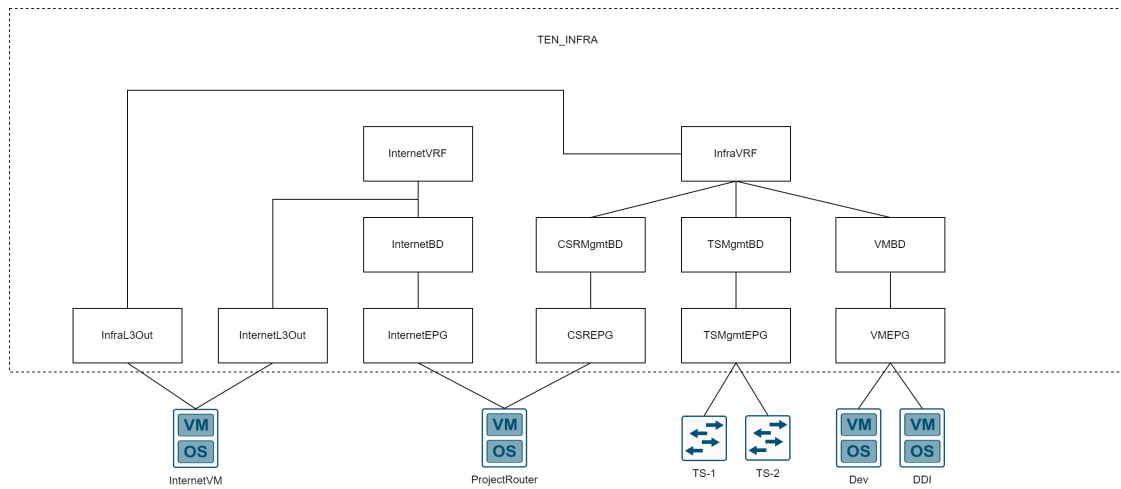
Figure 5.5: ACI Policy Overview

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 pur-
pose 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 REST-
    CONF 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   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 6.1 shows how AS 65001 was chosen.

Figure 6.1: 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 6.2 shows the configuration of the OOB IP addresses.

Configured Nodes

Node ID	Name	IPv4 Address	IPv4 Gateway	IPv6 Address	IPv6 Gateway
1	APIC	192.168.0.125	192.168.0.254	fe80::522f:a8ff:fe6a:d1e0	2001:420:28e:2020:acc:68ff:fe...
101	LEAF101	192.168.0.18	192.168.0.254	::	::
102	LEAF102	192.168.0.19	192.168.0.254	::	::
103	SPN103	192.168.0.16	192.168.0.254	::	::

Figure 6.2: ACI Out of Band IP Configuration

6.1.1 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 6.3 shows the configuration of the VMWare integration.

Properties

Name:	ACI-DVS
Virtual Switch:	Distributed Switch
Associated Attachable Entity Profiles:	Name
	aepVMM

Encapsulation:	vlan
Configure Infra Port Groups:	☐ To configure port groups for virtual apic
Delimiter:	
Enable Tag Collection:	☐
Enable VM folder Data Retrieval (Beta):	☐
Access Mode:	Read Only Mode	Read Write Mode
Endpoint Retention Time (seconds):	0
VLAN Pool:	vlanVMM(dynamic)
Security Domains:	🗑 +

Name	Description
No Security Domains Discovered	

vCenter Credentials: 🗑 +

Profile Name	Username	Description
VC	administrator@vsphere.l...	

vCenter: 🗑 +

Name	Type	IP/Hostname	Associated Credential
VC	vCenter	192.168.0.128	VC

Figure 6.3: 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.

Properties					
Description: optional					
Pairing Type: explicit					
Explicit VPC Protection Groups:					🗑 +
▲ Name	Domain Policy	Switches	Logical Pair ID	Virtual IP	
LEA101_102_VPC	VPCPol	101, 102	1	10.254.112.67/32	

Figure 6.4: 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.

Properties			
Name: VPC			
Description: optional			
Link Aggregation Type: Port Channel (PC) / Virtual Port Channel (VPC)			
Attached Entity Profile: aepVMM		Egress Data Plane Policing:	select a value
CDP Policy: system-cdp-enabled		Fibre Channel Interface Policy:	select a value
Link Level Policy: linkAuto		Ingress Data Plane Policing:	select a value
LLDP Policy: select a value		L2 Interface Policy:	select a value
Port Channel Policy: system-lacp-active		Link Flap Policy:	select a value
CoPP Policy: select a value		Link Level Flow Control:	select a value

Figure 6.5: 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 6.6.

Name: LEA101_102			
Description: optional			
Alias:			
Interface Selectors:			🗑 +
▲ Name	Blocks		Policy Group
ESXI-Host	1/2		VPC

Figure 6.6: vPC Interface Assignment

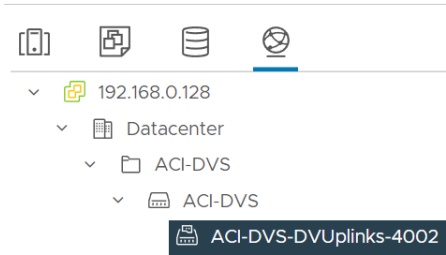In vCenter, figure 6.7 shows that ACI has correctly pushed the DVS to vCenter.

Figure 6.7: vCenter Distributed Virtual Switch

Now the uplinks need to be assigned to the LACP group that has been created by ACI. Figure 6.8 shows the configuration of the uplinks to use LACP.

Figure 6.8: 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 6.9: vPC Status on Leaf 101

6.1.2   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 6.10 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 6.10: Infra L3Out Configuration

Figure 6.11: 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 6.12 shows the configuration of the L3Out interface profiles.

Figure 6.12: 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 6.13: 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 6.14: 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.

6.2   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.

6.3   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.

6.3.1   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.

6.3.2   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 6.15 is the rackspace layout feature with a label used to illustrate the name of the
row.

Figure 6.15: 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 6.16 is the utility panel where racks and labels can be dragged and dropped onto the rackspace representation.
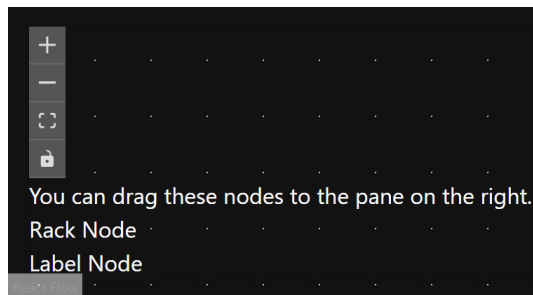

Figure 6.16: Drag and Drop

6.3.3   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 6.17 shows the modal with the interface profile mapping form.

Figure 6.17: 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 6.18 shows the selection dropdown.
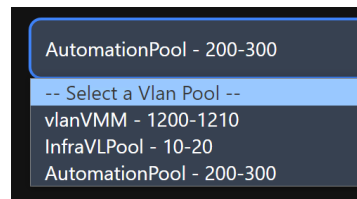

Figure 6.18: VLAN Pool Selection

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.

6.3.4 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 6.19.
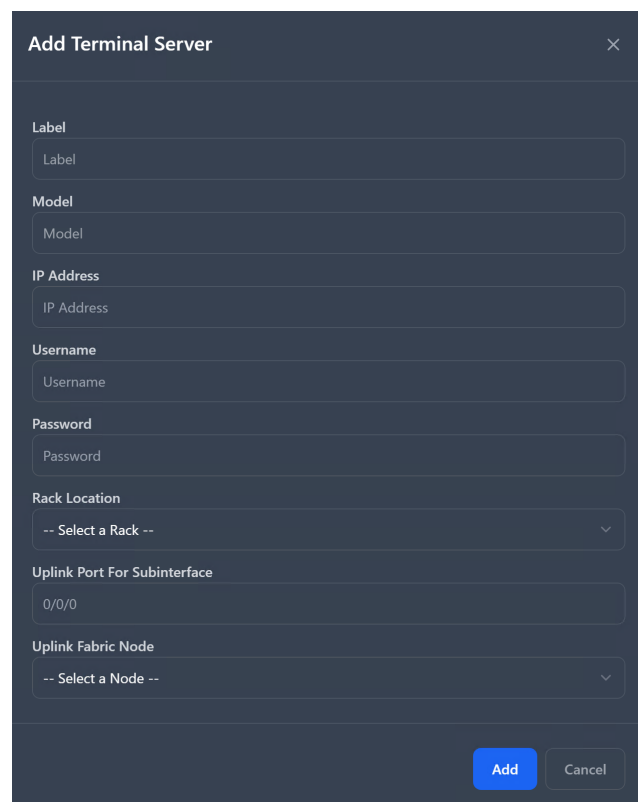
Figure 6.19: Terminal Server Addition Form

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.

6.3.5 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 6.20 shows the popover that appears when hovering over a rack.
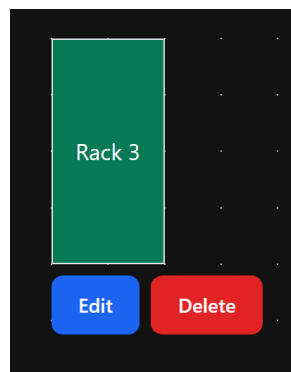
Figure 6.20: Edit Rack Popover

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 6.21.

Figure 6.21: 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.

6.3.6   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 6.22 is the modal form section that allows the user to select the racks that the project will occupy.

Figure 6.22: 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 6.23: 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.

6.3.7   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.

```php
{
    $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 6.24: Project Creation Job

Figure 6.24 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 6.25 is the job that is dispatched from the create project job, which will in turn provision the newly cloned project router.

```php
{
$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 6.25: 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 6.26. 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 6.26: 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

7.0.1 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 2 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 shows the successful ping test, showing that the ACI fabric is being provisioned correctly.

Figure 7.2: Test VM 1 pinging Test VM 2

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.

Figure 7.4: Test VM 1 pinging the internet

7.0.2 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.

```php
$project = Project :: with ('racks.terminalServer', 'vlan')->find ($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

7.0.3   Multiple Projects

To test intra-project communication is not possible, so the original test project was re-
duced 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 re-
spectively 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.

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.

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.

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

7.0.4   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.
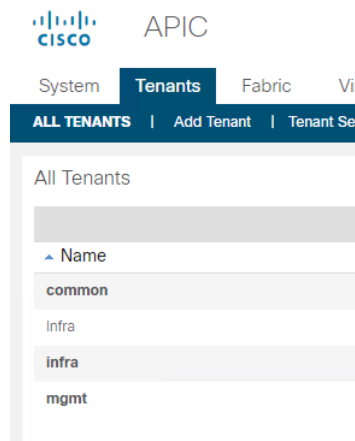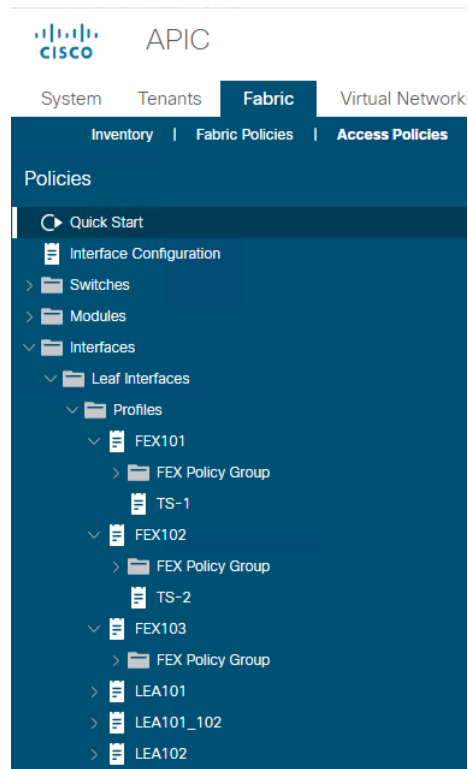

Figure 7.12: ACI Fabric Tenants
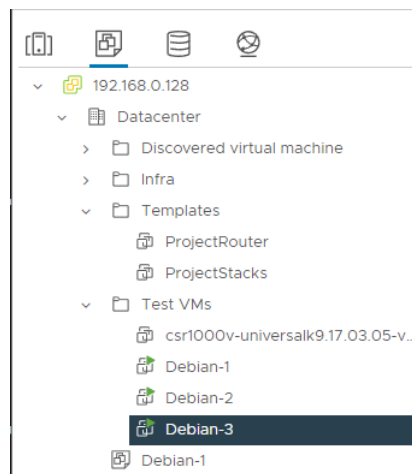
Figure 7.13: ACI Fabric Interface Policies

Figure 7.14: vCenter Inventory

Chapter 8

Evaluation

Chapter 9

Conclusion

References

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/

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.

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

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

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

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