Testing of SD-WAN Vendors APIs For Service Providers Integration

Irfan Azhar

A thesis submitted to Auckland University of Technology
in partial fulfilment of the requirements for the degree of
Master of Information Security and Digital Forensics (MISDF)

2021

School of Engineering, Computer and Mathematical Sciences

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning, except where due acknowledgement is made in the acknowledgments.

Signature of student

(11 June 2021)

Acknowledgements

Firstly, I would like to thank God for giving me this opportunity to complete another step in academic life.

I sincerely thank my primary supervisor, Professor Edmund Lai, for his tremendous support for this thesis. He helped me to get this research completed successfully. His supervision helped me to correctly apply the research principles to this thesis and he fully supported this thesis from just an idea in the start to fully completed research work and thesis in the end. I want to acknowledge and thank him for his helpful guidance and valuable feedback throughout the completion of this thesis after each progress report of the thesis.

I would also like to thank my employer Spark New Zealand who fully supported financially for this research as well as supported by allowing me to spent time to complete this study.

I would also like to express my gratitude to the vibrant community of AUT – lecturers, and classmates.

This thesis would not have been completed without full support of my family specifically my kids who tolerated hours of my daily work in front of laptop completing this research.

Abstract

Wide area network (WAN) is on the verge of another technology transformation where most of the WAN service providers are either adopting Software defined WAN (SD-WAN) or are planning to adopt it. SD-WAN integration with the existing fulfil, assurance and billing (FAB) stack of the service provider using application programmable interfaces (APIs) will be critical in SD-WAN adoption and selection of a suitable SD-WAN vendor. There are existing models to compare the SD-WAN vendors, but they overlook the APIs and do not compare the APIs provided by these vendors.

This thesis examines the existing frameworks for evaluating APIs, adapts them to define a framework for evaluating the SD-WAN vendor-provided APIs, applies the framework to two leading SD-WAN vendors' solutions and performs a comparative analysis of their APIs from the solution integrator's or architect's perspective.

The framework developed in this thesis, consisting of eight different aspects of APIs that needed to be tested to compare the SD-WAN vendor's APIs capabilities. As part of the framework, a structure is defined to score these aspects and allowing capability and flexibility to researcher to assign different weightage to each aspect based on the researcher requirements. The research conclude that the framework developed in this thesis allow solution architects to compare the vendors APIs and demonstrated the application of framework on two SD-WAN vendors.

Table of Content

Table of Figures

List of Tables

Glossary of Terms

API	Application Programming Interface
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IP	Internet Protocol
IT	Information Technology
LAN	Local Area Network
LTE	Long Term Evolution
MPLS	Multi-Protocol Label Switching
NMS	Network Management Systems
S	Score
SDN	Software Defined Networking
SD-WAN	Software Defined Wide Area Network
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
V_1	Cisco Meraki
V_2	VMware VeloCloud
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
W	Weightage
WAN	Wide Area Network

1.0 Introduction

This chapter provides introduction, background, and objectives of this thesis. It includes historical evaluation of the WAN, SD-WAN, importance of API for SD-WAN and basic concepts important to understand this research. The chapter also cover the motivation of the researcher to conduct this research, what are the aim and objective of this research and finally list down organization of the entire thesis.

SD-WAN technology is the new addition in the traditional WAN component of enterprise technology where there has been not much advancement in last 15 to 20 years. If we compare traditional WAN with IT system field, systems have been through a major transformation, where systems are no more physical but virtual and now mostly being deployed in the cloud. Enterprises do not need to care about underlaying platforms and technology to deploy these servers, where servers are updated automatically, and servers can be deployed across the globe with click of a button. This software defined abstraction has helped to hide all the complexities as a result IT system administration has evolved rapidly. With SD-WAN, same principle of Software defined networking (SDN) is being applied to traditional enterprises Wide Area Networking (WAN).

Software defined wide area network (SD-WAN) is predicted to be worth around $26 billion of the telecommunications market by 2026[2]. Most of the telecom service providers are either adopting SD-WAN or are planning to adopt it soon. There are many vendors who are providing SD-WAN technology and one of the challenges for Service providers is to evaluate these vendors to find out which vendor will be best fitted for their business. There will be many factors involved in evaluation of a SD-WAN vendor where one of the key factors will be vendor's SD-WAN technology integration with the existing fulfil, assurance and billing (FAB) stack of the service provider. SD-WAN vendors mainly uses application programmable interfaces (APIs) for such integrations. This thesis is designed to examine such APIs provided by the leading SD-WAN solution vendors and provide a comparative analysis on their usefulness from the solution integrator / architect perspective.

SD-WAN also provide the concepts of underlays and overlays[3]. Underlays are traditional WAN technologies like, MPLS circuit, broadband internet, and any mean of connectivity even the use of wireless connectivity like LTE and satellite connection is also consider types of underlay. On the other hand, Overlays is using software to create layer of virtual network abstraction that is built on top of these underlay network infrastructure using tunnels. Another concept of SD-WAN is called SD-WAN fabric. SD-WAN fabric compromise of all the SD-WAN nodes connected using overlay tunnels.

Controller is another core component of SD-WAN which provide orchestration and control the overlay tunnels and traffic routed on these overlay tunnels by each SD-WAN node in the SD-WAN fabric. Controller can be a single node providing all the functionality of orchestrator, controller and user portal or these functions can be divided into different nodes. For this thesis controller mean single node providing all the functionality of controller, orchestrator and same node is providing API interface to connect to entities outside SD-WAN fabric.

SD-WAN controller use APIs to communicate with the external entities.

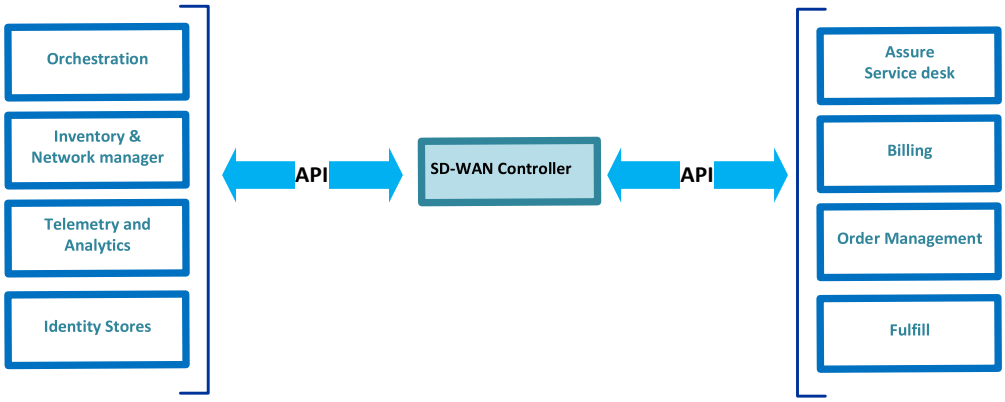


Figure 1: SD-WAN use of APIs to integrate with external entities

These controllers may also provide traditional interfaces like, SNMP, Syslog and NetFlow but these traditional interfaces are for backward compatibility as some of the service providers or enterprises may not be ready to consume the API. Regardless of these traditional interfaces to provide the backward compatibility, the way forward for SD-WAN integration and preferred way to integrate with external entities is APIs. With service provider adopting SD-WAN, integration of the technology with the FAB stack using API will be one of the key aspects for adopting a specific vendor over another.

SD-WAN apply the concept of Software Define Networking (SDN) on traditional WAN technologies [4] like MPLS. The main difference in SD-WAN and traditional WAN is separation of control plane and data plane where control plane is managed by a controller node or simply called controller.

1.1 Motivation

This section will discuss the motivations to develop this study and why APIs of SD-WAN vendors are focus of this research.

The main motivation of the author for this thesis came from several year of working for service provider as architect, particularly on WAN technologies. The author observed that there are multiple vendors providing SD-WAN technology and claim to be better than their competitors and there are different evaluation frameworks available for comparing SD-WAN vendors but none of these focus on SD-WAN APIs. While there are several different articles written to highlight the importance of APIs for SD-WAN vendors and there are already multiple frameworks defined for testing of the APIs but none of these frameworks have been applied to SD-WAN vendor provided APIs.

This is turning point in the WAN technology and after 15 years there is new transformation where WAN is moving towards post MPLS era. While all the service provider and technology vendors are willing to jump into this transformation, right combination of vendor and service provider is key for successful adoption of the technology where APIs will play key role for successful adaptation.

For any large service provider, adoption of any new technology like SD-WAN is highly dependent on its integration with service provider existing fulfil, assure and billing (FAB) stack. A well-integrated solution will allow service provider to automate many tasks including provisioning, configuration changes and billing to customer without any manual intervention.

The researcher wants to ensure that there is a framework available that can be used by any solution architect or integrator to evaluate the API capability of any SD-WAN vendor. Although the research is specifically around SD-WAN vendor provide API, the researcher goal is to keep the framework generic enough to be adopted for evaluate and test any API provided by any vendor.

1.2 Aims and Objectives

The main aim of the thesis is to define a framework to evaluate and compare the API capabilities of leading SD-WAN vendors.

The first objective will be to evaluate existing API testing frameworks and to research if these frameworks are applicable for testing of SD-WAN vendors APIs. While there are already multiple frameworks defined for testing of the APIs and but none of these frameworks have been applied to SD-WAN vendor provided APIs.

The second objective of the research will be to where possible enhance existing framework to ensure it cover all the possible aspects relevant to SD-WAN APIs and to ensure that framework used to evaluate the APIs is focusing on aspect of its integration with service provider existing FAB stack.

The third objective of the research will be to apply the framework on two leading SD-WAN vendors and to generate results and to do the comparative analysis on the results to validate the application of the framework. The aim will be to define the framework in such a way that any telco architect will be able to use this framework to do comparative analysis of any two or more SD-WAN vendors.

1.3 Organization

This thesis is divided into six chapters.

The first chapter is introduction which provide introduction, background, and objectives of the thesis. The introduction chapter cover the historical evaluation of the WAN, Why SD-WAN, importance of API for SD-WAN and basic concepts important to understand the aim and objective of the research. The chapter also cover the Motivation of the researcher to conduct this research, what are the aim and objective of this research and finally list down organization of the entire thesis.

The second chapter cover the existing research work done on defining the framework for API comparisons and review these frameworks. Identifying the research and limitation of existing literature work. The literature review done in this chapter will be used as foundation of this research. The chapter two conclude with the summary of research goals.

Third chapter cover the research methodology and definition of the framework. This chapter start with identification of aspects that needed to be included in API testing framework. What aspects are already covered by existing research and what are added by this research. The chapter then list down the main research question and other two related questions covered by this research. The third chapter then provide detail of each aspect of testing framework, why each aspect is included and what part of framework it covers. Third Chapter also cover the testing methodology, define the variables and equations used in this research. The chapter also cover the vendor selection criteria and then cover the data acquisition methods, variables and their values required for data acquisition, tools and methods of data acquisition and data format in which data will be collected. The chapter then collect all the data required to conduct this research and list down the details of collected data for each aspect of the testing framework.

Forth chapter reports the finding and results of the research by apply framework on two leading SD-WAN vendors. The chapter apply each aspect and list down the result of each aspect and provide both vendor score for each aspect of testing framework along with the reasoning of assigned score and calculate weighted score by applying weightage of each aspect to the score given to each vendor. The chapter provide deep analysis of the results

Fifth chapter discusses the research findings and if the research questions have been answered. The discussion is based on results in chapter four. Chapter further discuss each aspect and their scores and, in the end, discuss the overall score of the vendors.

Sixth and final chapter provide the overall review of the thesis, what contribution this thesis provides in the field of SD-WAN and API comparisons, what are the limitation of the research work and what future research is required to further advancement in this research.

2.0 Literature Review

This chapter cover the existing research work done on defining the framework for API comparisons and review these frameworks. This chapter also identify the existing literature works and limitation of existing literature works. The literature review in this chapter will be used as foundation of this research. This chapter conclude with the summary of research goals.

There are large number of research available in the field of SDN and SD-WAN. This chapter presents to the reader about what is already known and help to understand the research already done in the field of SD-WAN and APIs. This chapter will also set the theoretical foundation and will review the concepts, definitions and theories present in already presented research and studies.

2.1 Introduction

SD-WAN is combination of existing technologies like SDN, automatic failover of underlays, visibility and usage reporting of these underlays and application awareness. One of the concepts of SD-WAN is use of API instead of relying of using Command Line Interface (CLI) or Graphical User Interface (GUI). The importance of the API is evident from the existing research on SD-WAN.

2.2 Related Work

SD-WAN vendors provided APIs will play key role for such integration. Not All APIs are equivalent as quality APIs are rare and minimal flaw at API design time could lead to major constraint in later stages as APIs are designed once but used numerous times. The cost of flawed APIs is exponential as the main purpose of APIs is to create an abstraction layer and hide details where bad APIs need detail working and programmers required more time to work with poorly designed APIs[5].

API in one of the seven keys to unleashing the full potential of an SD-WAN system and is considered key for automation and seamless integration with existing tools[6]. APIs are used for digital integration with enterprise customers and as basis of ecosystem partner integration. Not only for service providers but also for service provider customers such as large enterprises, APIs importance and relevance specifically to their business is critical[6]. Service providers can open the APIs to their partner and customers for close integration and for flow of information from one system to another [7]

There are existing well defined strategy and frameworks for general API testing. These frameworks define the API development standards and list down considerations that should be taken into development and testing of an API. The API testing framework is applied to compare APIs of AWS and Azure as cloud service provider[8].

While there are several multiple research works already done to highlight the importance of APIs for SD-WAN vendors [9],[8], [10], [11]and there are already multiple frameworks defined for testing of the APIs [8], [12], [13], [14].

The work done by Vijayakumar [8] serve as basis of the research as it lists down six aspects of API architecture. Although the research was done specifically for AWS and Azure it set the right basis on what aspects of API needed to be considered when designing a framework to test the APIs. Sections specific to Azure and AWS APIs like API gateway and serverless APIs are not included in this research as these are not relevant to SD-WAN.

The research [12] and [13] provide a framework for testing cloud-based APIs. These highlight the importance of API testing to gauge the stability and reliability of the API. The focus of work is to test different types of API and to create a testing framework.

The research [14] provide valuable inputs on how an API testing can be automated. It highlight the challenges and complexities involve in automating the API testing specific to sequencing the API call tests. The research propose the API testing tools and procedures those can be adopted for API testing.

None of the existing frameworks have been applied to SD-WAN vendor provided APIs. There is further research required first to find out if any of the existing API testing framework can directly be applied to SD-WAN vendor provided APIs and secondly by applying a framework a meaningful conclusion can be made on usefulness of specific vendor APIs compare to another vendor.

3.0 Research Design

This chapter cover the research methodology and definition of the framework. This chapter start with identification of aspects that needed to be included in API testing framework. What aspects are already covered by existing research and what are added by this research. The chapter then list down the main research question and other two related questions covered by this research. The chapter then provide detail of each aspect of testing framework and why each aspect is included and what part of framework it covers. The Chapter also cover the testing methodology and define the variable and equation used in this research. The chapter then cover the vendor selection criteria and the data acquisition methods, variables and their values required for data acquisition, list down tools and methods used of data acquisition and data format in which data will be collected. The chapter then collect all the data required to conduct this research and provide details of collected data for each aspect of the testing framework.

3.1 Introduction

In recent years, several studies have tried to define a framework for testing the APIs. The research is inspired by the API testing framework defined by [8] use six aspects of API architecture as base for testing framework.

These six aspects as identified by the author are:

- Security
- Performance
- Usage & Telemetry
- Developer Experience
- Integration
- Error Handling

The framework was specifically designed for AWS and Azure and focus on API gateways and serverless API applications. Both are not true for the case of SD-WAN vendors APIs as there is no API gateway nor did serverless API applications are relevant for SD-WAN.

The existing framework did not provide any comparison mechanism or did not used any rating systems to gauge one API vs the other.

3.2 Research Questions

The research questions are derived from literature review done in chapter two. Literature review list down the API testing framework available, but it was observed that these frameworks were not applied to test any SD-WAN vendor APIs. Existing frameworks were also not enriched or comprehensive enough to cover all the aspects of API testing for SD-WAN integration with service provider FAB stack.

Following are the research questions which will be answered by this thesis:

Q1 Could the existing framework as defined in API architecture and development [15] can be directly applied to test the usefulness of SD-WAN APIs for integration with service provider existing stack?

Q2 In what ways are leading SD-WAN vendor's APIs alike?

Q3 How could the competence of an SD-WAN vendor's API be evaluated in comparison with another vendor?

3.3 Design of study

Existing framework provides aspects which needed to be gauge but provide no mechanism to rate these aspects. Furthermore, two more aspects which are key to rate the SD-WAN APIs needed to be included these are:

- Documentation
- Online community

This bring the total aspects to eight different aspects these are:

- Security
- Performance
- Integration
- Usage & Telemetry
- Developer Experience
- Error Handling
- Community
- Documentation

Following section provide brief introduction of each aspect and rational of adding two more aspects of Documentation and Online Community to the framework

Security

API security is the protection of the overall integrity of the APIs. API security not only cover the traditional security that is authentication and authorization to use but also cover the granular data exposure based on role and access level. The security measure includes protection of API interface itself by rate limiting the number of calls, logging the usage of each API call and key used for that call.

Performance

Performance of an API is primarily related to responsiveness of the API against API calls. Other aspects of APIs which come under performance are availability and caching of the API calls. Caching is a mechanism to have data readily available for repetitive API calls.

Integration

API integrations relate to backward compatibility of the APIs. APIs are entities which goes through continuous improvement and changes. With every update how API developers ensure

that the integrated solution will keep on working without any issues and update to the API will not have any impact to existing integrations. Usually, integration is controlled by major and minor version numbers of the APIs where there can be major changes in the major version, but the interface will keep responding to calls tagged with previous version till previous version is discontinued after providing ample amount of time to integrated systems to upgrade to newer version of API.

Usage & Telemetry

Usage and telemetry provide the stats on the usage of APIs. These stats are useful for API developer to focus on the API calls which are consumed most. Data insight using telemetry guide the developers in better understanding of API adoption and usage. The telemetry can also be about the client of the API for instance who, how and how many calls are made by specific source etc.

Developer Experience

The API interface usage depends upon the developer experience specifically how straightforward for developer was to use the API. The Developer experience cover many aspects of the design but mainly on documentation, help available from vendor and online development portal and community.

Error Handling

Some of the frustration on the API for developers and integrators is on error handling of the API calls. API calls can fail for number of reasons. Each failed API call should be handled in a way that it provides error description and number every different reason for an attempt to fail should be responded by different error number. Error numbers and how to guide to resolve this error should be maintained in the vendor documentation.

Community

Community aspect is added to the framework by this research. Community aspect is critical as it serve as bridge between the API provider, experienced users and new adaptors of the API. For community flourishing vendor usually provide an online portal serving as home ground for API integrators. This portal is usually open for anyone to join where initially most of the post are from APIs developers themselves but as community grow, users after getting some experience in using the API those start to answer new integrator queries. Vendors can fast track the community development by providing incentives to the contributors and encouraging them participate as much as possible. An active community help to improve the quality of the APIs, identify the bugs and performance issues and requesting new features. The active community members are usually also early adopters of the new features which can be released to controlled group of active community member for testing and identifying any bugs in the new release of the API.

Documentation

Documentation is 2nd aspect that has been added by this research as documentation can be one of the most important factors in the API. Not only documentation is key but complete and accurate documentation touch all the other aspects of the API development including error handing, community, versioning. The developer experience for example is mainly based on

comprehensiveness and accuracy of the documentation. A well-documented API will be easy to adopt, and detail documentation answer most of the potential queries of the API users.

3.4 Testing Methodology

Although existing framework provide details around what are the key aspects of API that needed to gauge but this framework did not provide any comparison mechanism or did not used any rating systems to gauge one API vs the other.

All the 8 aspects that will be tested will be denoted by A1, A2…., A8

This testing will use two variables one to gauge the value scored by specific vendor in that specific aspect. That variable will be denoted by S in this testing. The value of S will be given in any number from 1-10 based on testing results.

2nd variable used in this testing will be the Weightage of each aspect from the tester perspective. The weight variable will be denoted by W in this testing, and it can be valued between 0-1. The weightage variable is to give the flexibility to the tester in framework to change the weightage of any variable that is not critical for their integration service provider elements as what each service provider use, and integration entities may be different.

Finally, each vendor will be denoted by V. For this testing two vendors will be tested V1 and V2. Final score of V will be as follow:

SV_1 = Sum($A_1(S_{11}*W_1)$, $A2(S_{12}*W_2)$…….$A_8(S_{18}*W_8)$)

SV_2 = Sum($A_1(S_{21}*W_1)$, $A2(S_{22}*W_2)$…….$A_8(S_{28}*W_8)$)

Although the tester can choose any value of Weightage (W) for specific aspect, but the same value will be applied for each vendor where the scored (S) value will be based on that vendor score in that aspect. Both vendors needed to be tested for same number of aspects. For example, in this testing 8 aspects has been defined in the framework.

For this testing two vendors will be selected and following table will be populated during the testing

Aspects	Weightage	Vendor1 Score	Vendor1 Weighted Score	Vendor2 Score	Vendor2 Weighted Score
A1 – Security	W_1	S_{11}	W1*S11	S_{21}	W1*S21
A2 – Performance	W2	S_{12}	W2*S12	S_{22}	W2*S22
A3 – Integration	W3	S_{13}	W3*S13	S_{23}	W3*S23
A4 – Usage & Telemetry	W4	S_{14}	W4*S14	S_{24}	W4*S24

A5 – Developer Experience	W5	S_{15}	W5*S15	S_{25}	W5*S25
A6 – Error Handling	W6	S_{16}	W6*S16	S_{26}	W6*S26
A7 – Community	W7	S_{17}	W7*S17	S_{27}	W7*S27
A8 – Documentation	W8	S_{18}	W8*S18	S_{28}	W8*S28
Total			SV_1		SV_2

Table 1: Vendor Score Testing Methodology

Vendor Selection

Vendor selection for any specific ISP or enterprise can be based on multiple reasons. This research doesn't cover the full vendor selection criteria for SD-WAN adoption but only the API testing of short-listed vendors to do comparative analysis of those vendors API capabilities.

This research will focus on two SD-WAN vendors from the leaders quadrant as per [16] Gartner magic quadrant and eweek top SD-WAN providers [17] two leading vendors for SD-WAN are Cisco and VMware.

Figure 2: Gartner magic quadrant 2020 for SD-WAN vendors [15]

These vendors will be denoted by V1 and V2 throughout this research:

V1 = Cisco – Meraki Solution

V2 = VMware – VeloCloud Solution

Although only two vendors will be tested in this research, the framework used in this research for comparing SD-WAN APIs is equally applicable to any other SD-WAN vendor.

3.5 Data Acquisition

This section covers how the data is collected from the controller for the analysis.

The data was collected using Google Postman tool. The free version of the tool was used for this data collection. Data Collector is the person or who collected the data. Data collected used Postman as API client on a laptop and queried to the orchestrator of the SD-WAN vendor residing on the remote cloud location. SD-WAN orchestrator or controller was responsible to provide API interface. Connectivity from Laptop to SD-WAN controller will be via https TCP/443 port.

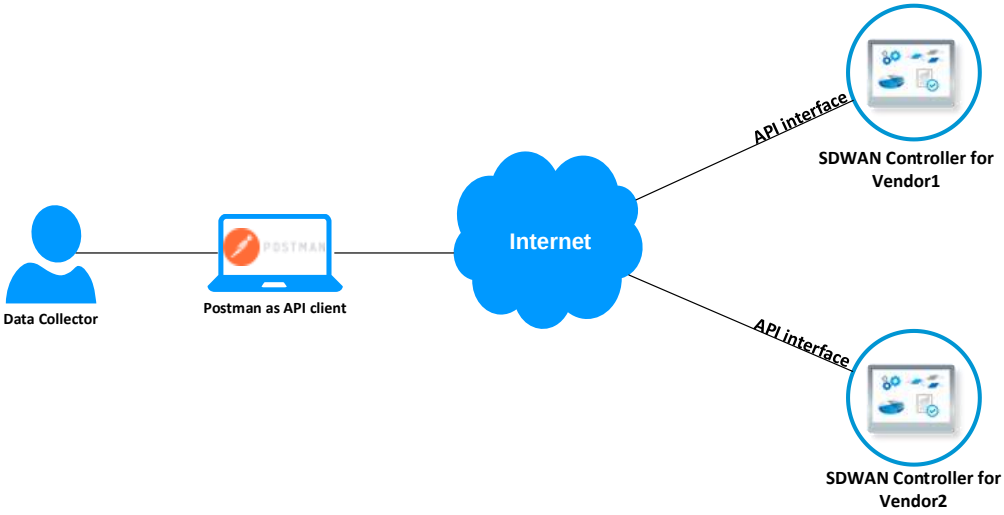


Figure 3: Data Collection from SD-WAN controllers using Postman

To query data on API interface the data collector at minimum required to know the URL or IP address of the controller usually denoted by baseURL. Some additional bare minimum variables can be API Key for authentication and OrganizationID. As these controllers are usually multi-tenant and Organization ID is unique number which identify the organization of that user.

Variable	Type	V1	V2	Description
baseURL	Mandatory	https://dashboard.meraki.com/api/v0	https://vco301-syd1.velocloud.net	Controller address on the internet. It can be a web address or can be as simple as an IP address.
Authentication Type	Optional	APIKey	Username/Password	Unique key to authenticate the request. It will depend on the type of (if any) authentication controller is doing. Some controllers may use a username and password for authentication.
organizationID	Optional	Use GET request to get the list of organizations	Use POST request to get the list of organizations	Identify the organization on the controller specifically when the controller is multi-tenant and hosting multiple organizations

Table 2: Data Collection variables and their values

SD-WAN vendors usually have multiple instances of the controller and baseURL point to the SD-WAN controller instance where API user can connect usually using the APIKey. If the controller is multitenant controller, an ISP usually have multiple organization representing

separate customers and organizationID is either provide by the vendor or ISP can get the list organizations with orgainziationIDs using an API request. Some SD-WAN vendors allow to deploy local instance of the controller and to find the baseURL of the local instance for the API call ISP may need to refer the documentation provided by the vendor.

Note that to get access to API interface of SD-WAN vendor controller an ISP need to have an account on SD-WAN partner portal which allow them to create the organizations. Different vendors have different requirements to get this access and usually SD-WAN vendor account manager for the ISP is first point of contact to get started.

API Calls

Another aspect of the API integration is what API calls can be made to the controller and what method is required for each API calls and lastly what is expected in response of the API calls.

API calls can use different communication protocol but rest APIs which use https TCP/443 for communication has been adopted as de facto standard for the API calls. This thesis will be using Rest API calls for the collection of the data.

Methods for API calls

There are five different methods an API can use for calling API to perform suitable actions. These calls are listed below.

- GET
- POST
- PUT
- DELETE
- PATCH

GET

GET is used to retrieve resources information from the server using API interface. The main property of the GET calls is that the GET request should be idempotent that is making same call using same variable should always return the same data unless data has been changed using a POST/PUT/PATCH request.

POST

POST is used to create a new resource using API interface. POST request is not idempotent that is making same call using same variable can result in creating two different resources. POST resource can use to create a collection of resources using a single call.

PUT

PUT is used to update a resource (If resource doesn't exist the same API call can also create the request within the same call. PUT request is used to either update or create a single resource using call.

DELETE

DELETE is used to delete resources. Delete operation are also idempotent that is resource is deleted using the request. Repeated request to delete same resource will result in a response code of resource not found.

PATCH

PATCH is used to partially update a resource. Different between PATCH and PUT is when using PUT you update an entire resource while PATCH partially change the content of the resource.

Response Code

The API interface of the server response to the API request using codes to inform the client on status of the response. These response codes are used to convey the results of the client's request. Below are five different categories of response code:

1. 1xx: Informational
2. 2xx: Success
3. 3xx: Redirection
4. 4xx: Client Error
5. 5xx: Server Error

The most commonly known response is 2xx and 4xx for example 200 [OK] which indicate the request was successful while 400 [Bad Request] which indicate server didn't understood the request.

Below is an example of the GET request which use baseURL variable and request call to get the organization details

Figure 4: Get request example using Postman

API calls use the key value pairs in the header to pass different fields to server for example in below this request key parameter is passed to authenticate to the API interface and return content is requested in JSON format.

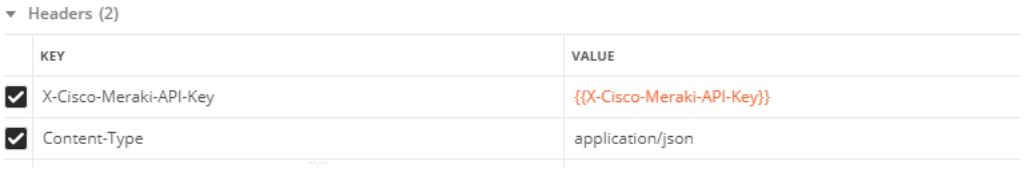


Figure 5: Header Key Value pairs example using Postman

Below is an example of API response code of 200 which indicate the request was successful.

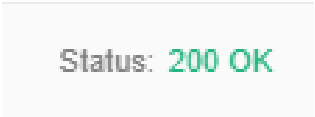


Figure 6: Response Code example using postman

Data format

Another concept related to API calls is data format. There is different type of data formats for example: plaintext, XML, YAML and JSON. Usually, API return data is XML or JSON format where JSON is becoming de facto standard for of data format. JSON data consist of collections of pairs of name/value and ordered lists. Here is an example of JSON data

```
1. [
2.     {
3.         "id": "XXXXX",
4.         "name": "XXXXX",
5.         "url":
   "https://n215.meraki.com/o/gn8aWc/manage/organization/overview",
6.         "samlConsumerUrl": null,
7.         "samlConsumerUrls": []
8.     },
9.     {
10.          "id": "XXXXX",
11.          "name": "XXXXXX",
12.          "url":
   "https://n69.meraki.com/o/3NDVPdfb/manage/organization/overview"
   ,
13.          "samlConsumerUrl": [],
14.
15.      }
16.  ]
17.
```

Same data in XML will be represented as below

```
1. <?xml version="1.0" encoding="UTF-8" ?>
2. <root>
3.   <row>
4.     <id>XXXX</id>
5.     <name>XXXXX</name>
6.
   <url>https://n215.meraki.com/o/gn8aWc/manage/organization/overvi
   ew</url>
7.     <samlConsumerUrl/>
8.     <samlConsumerUrls/>
9.   </row>
10.    <row>
```

```
11.      <id>XXXXX</id>
12.      <name>XXXXX</name>
13.
   <url>https://n69.meraki.com/o/3NDVPdfb/manage/organization/overv
   iew</url>
14.      <samlConsumerUrl/>
15.      <samlConsumerUrls/>
16.    </row>
17.  </root>
```

For this research JSON data format will be used for collecting and representing collected data.

3.6 Data Collection

Apart from Postman client tool which will be used to interact with API interface some aspects of the framework for example community required different way to collect the data. Following section list down each aspect of the framework and how data was collected for that aspect.

Security

Postman was used to check the authentication and authorization method of the API. Vendor documentation will be used to find out if there is any rate limiting implemented for number of requests in a given time, amount of data that is collected and audit logging of the API usage.

V1A1 Security – Cisco Meraki

Use API Key for authorization and API key is passed in the header of each request.

Here is content of GET request console output of a Postman API request to Cisco Meraki portal API interface.

```
1. GET /api/v0/organizations
2. X-Cisco-Meraki-API-Key: XXXXXXXXXXXX
3. Content-Type: application/json
4. User-Agent: PostmanRuntime/7.15.0
5. Accept: */*
6. Cache-Control: no-cache
7. Postman-Token: XXXX
8. accept-encoding: gzip, deflate
9. referer: https://dashboard.meraki.com/api/v0/organizations
10.  Connection: keep-alive
11.  HTTP/1.1 200
12.  status: 200
13.  Server: nginx
14.  Date: Tue, 18 Aug 2020 10:49:31 GMT
15.  Content-Type: application/json; charset=utf-8
16.  Transfer-Encoding: chunked
17.  Connection: keep-alive
18.  Vary: Accept-Encoding
19.  Cache-Control: no-cache, no-store, max-age=0, must-revalidate
20.  Pragma: no-cache
21.  Expires: Fri, 01 Jan 1990 00:00:00 GMT
22.  X-Frame-Options: sameorigin
23.  X-Robots-Tag: none
24.  Sunset: 2022-02-05T23:59:59Z
```

```
25.  Deprecation: 2020-08-05T23:59:59Z
26.  Link: <https://developer.cisco.com/meraki/api-v1/#!versioning>;
     rel="deprecation"; type="text/html",
     <https://api.meraki.com/api/v1/organizations>; rel="successor-
     version"; type="application/json"
27.  X-UA-Compatible: IE=Edge,chrome=1
28.  X-Request-Id: XXXX
29.  X-Runtime: 0.230725
30.  X-XSS-Protection: 1; mode=block
31.  Content-Encoding: gzip
32.  [{"id":"XXXX","name":" XXXX
     ","url":"https://n215.meraki.com/o/gn8aWc/manage/organization/ov
     erview","samlConsumerUrl":null,"samlConsumerUrls":[]},{"id":"XXX
     X","name":" XXXX
     ","url":"https://n69.meraki.com/o/3NDVPdfb/manage/organization/o
     verview","samlConsumerUrl":"https://n69.meraki.com/saml/login/3N
     DVPdfb/ADoQidpbkcdb","samlConsumerUrls":null]}]
```

Some observations in the above code:

- Line Number 2: API Key send to server for authorization
- Line Number 9: baseURL of the server API interface
- Line Number 12: Server response as 200 OK
- Line Number 32: Server response body in JSON format

Note that any identification, password and keys has been masked in all the examples and has been replaced with XXXXX.

Restricting key validity to specific source IPs

The further restrict the key validity the Meraki provide optional parameter to restrict the API calls to be accepted from certain listed IP addresses.

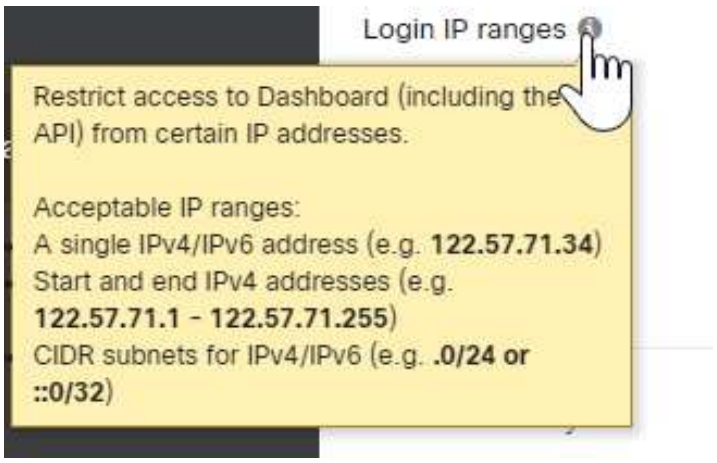


Figure 7: Cisco Meraki API Source API restriction snapshot from Meraki Dashboard

This option adds to the security of the API, as now although the controller is reachable from public internet the attack vector is now limited to specific IP addresses, this parameter on one hand reduces the exposure to specific source Ips and secondly it helps in case of Key compromise, Key can only be used from these specific IP address till the compromised key is revoked and new key is generated.

Rate limiting and usage stats

Rate limiting data is collected from the online documentation of Cisco Meraki API. The API limit 5 request per seconds, per organization with additional burst of 5 seconds allowed that is maximum 15 request in two seconds. Another limit is based on client IP address on concurrency of the request from same source IP. In one time a single source IP can concurrently request up to 10 requests per IP address.

Usage stats are taken from Cisco Meraki online dashboard. The Cisco Meraki provide a usage stat based on the API key two stats which are provided are when was the Key created and last usage of the API key.

Figure 8: API Key example from Cisco Meraki Dashboard

V2A1 Security – VMware VeloCloud

Here is content of POST request to list down the organization has access to.

```
1. POST /portal/rest/enterprise/getEnterprise
2. Content-Type: application/json
3. Authorization: Token XXXXX
4. User-Agent: PostmanRuntime/7.15.0
5. Accept: */*
6. Cache-Control: no-cache
7. Postman-Token: XXXXX
8. Host: vco303-syd1.velocloud.net
9. accept-encoding: gzip, deflate
10. content-length: 38
11. Connection: keep-alive
12. { "id": 3, "enterpriseId": 3 }
13. HTTP/1.1 200
14. status: 200
15. Server: nginx
16. Date: Tue, 18 Aug 2020 11:06:00 GMT
17. Content-Type: application/json; charset=utf-8
18. Content-Length: 732
19. Connection: keep-alive
20. X-Powered-By: Express
21. Strict-Transport-Security: max-age=31536000; includeSubdomains;
22. X-Frame-Options: SAMEORIGIN
23. { "id": 3, "created": "2019-09-13T02:15:32.000Z", "networkId":
    1, "gatewayPoolId": 9, "alertsEnabled": 1}
```

Some observations in the above code:

- Line Number 2: API Token required in each POST request for authorization
- Line Number 8: baseURL of the server API interface

- Line Number 12: Request body to supply the id of the enterprise
- Line Number 14: Server response as 200 OK
- Line Number 23: Server response body in JSON format

Note that some of the keys and IDs has been replaced by XXXXX. VeloCloud is using POST request for getting the information which is against the basic design principle of the REST based API [18] where GET request is designed for requesting information about the resources while POST request is to update resource(s).

For authorization the key name is authorization while the key value must start with Keyword Token followed by Token value. This could have easily been improved by using authorization-Token as Key name and token as value only.

VeloCloud also supports only cookie-based authentication. For cookie-based authentication a client first required to call a POST request to baseURL/login/operatorLogin method where username and password is passed to server in the body of the request.

```
1. POST /portal/rest/login/operatorLogin
2. Content-Type: application/json
3. User-Agent: PostmanRuntime/7.15.0
4. Accept: */*
5. Cache-Control: no-cache
6. Postman-Token: b8641e50-8fa9-4802-9b8d-41864034673a
7. accept-encoding: gzip, deflate
8. referer: https://vco303-
   syd1.velocloud.net/portal/rest/login/operatorLogin
9. Connection: keep-alive
10.  { "username": XXXXX, "password": XXXXX }
11.  HTTP/1.1 200
12.  status: 200
13.  Server: nginx
14.  Date: Tue, 18 Aug 2020 11:20:47 GMT
15.  Content-Type: text/html
16.  Transfer-Encoding: chunked
17.  Connection: keep-alive
18.  Vary: Accept-Encoding
19.  Strict-Transport-Security: max-age=31536000; includeSubdomains;
20.  X-Frame-Options: SAMEORIGIN
21.  Content-Encoding: gzip
```

Some observations in the above code:

- Line Number 1: POST request for cookie-based authorization
- Line Number 8: baseURL of the server API interface
- Line Number 10: Request body to supply the username and password
- Line Number 14: Server response as 200 OK

Upon successful authentication, these methods produce a response with a Set-Cookie header, from which a velocloud.session cookie is parsed. The Velocloud uses this token (which must be passed in a Cookie header) to authenticate subsequent requests from the client. Session cookies expire after a period (24 hours, by default) and may be refreshed by invoking the cookie-based authentication request again.

The issue with the cookie-based authentication is the stateful nature of the authentication [19] that is record needed to be kept on both server and client. Cookie-based authentication is discouraged, and it is also a limitation for scaling as server need to keep record of each client session.

Rate limiting and usage stats

There was no rate-limiting related documentation for VeloCloud.

Usage stats are taken from VeloCloud Orchestrator online portal. The VeloCloud provide only token creation date but no stats on when token was last used is available via the portal. Below is the content of token related information available via VeloCloud Orchestrator portal.

```
[
  {
    "Created": "08/16/20, 23:25:45",
    "Expiration": "08/16/21, 23:25:45",
    "State": "ENABLED",
    "Downloaded": "08/19/20, 00:18:07",
    "Created By": "irfanazher@gmail.com",
    "Token Type": "OPERATOR",
    "Customer": "",
     "Created For": "irfanazher@gmail.com"
  }
]
```

Performance

Postman was used to gauge the performance and responsiveness of the API calls. Vendor documentation will be used to find out if caching mechanism is used to data readily available for repetitive API calls.

V1A2 Performance – Cisco Meraki

Postman clients indicate response time of less than a second for each query as highlighted in the snapshot below:

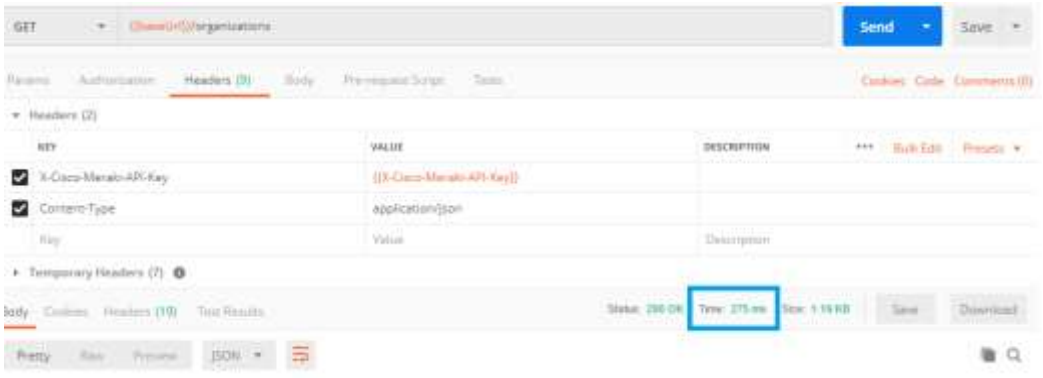


Figure 9: Cisco Meraki API request response time example using Postman

The response time also differ based on where your location and distance from the server location.

Cisco Meraki doesn't provide any caching mechanism but provide mechanism of paginated GET request where client can request a subset of information in the first request and subsequent information can be collected by changing provided parameter in next requests in order to get rest of the information. Here is an example where information is requested page wise:

```
<https://api.meraki.com/api/v0/networks/N_1234/bluetoothClients?perPage=5&startingAfter=0>; rel=first,
<https://api.meraki.com/api/v0/networks/N_1234/bluetoothClients?perPage=5&startingAfter=105>; rel=next,
<https://api.meraki.com/api/v0/networks/N_1234/bluetoothClients?perPage=5&endingBefore=0>; rel=last
```

Figure 10: Cisco Meraki paging mechanism example

V2A2 Performance – VMware VeloCloud

Postman clients indicate response time of less than a second for each query as highlighted in the snapshot below:

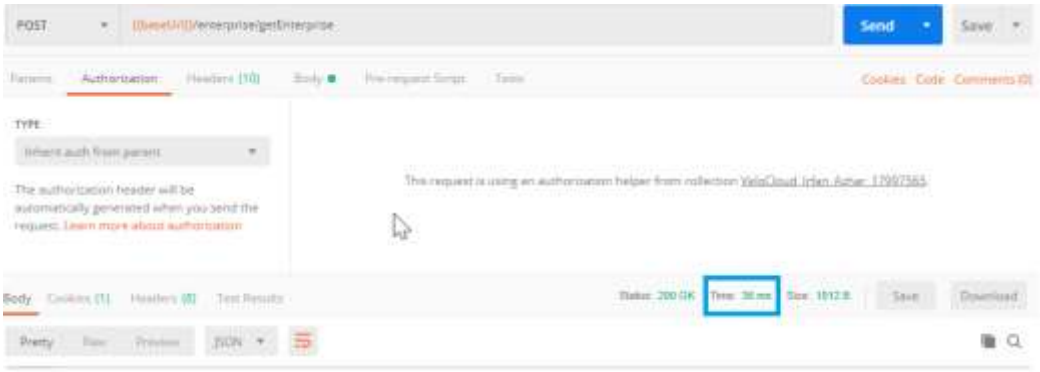


Figure 11: VMware VeloCloud API request response time example using Postman

The response time also differ based on where your location and distance from the server location.

VMware VeloCloud documentation does not cover any caching or pagination

Integration

Vendor documentation was used to check the version information.

V1A3 Integration – Cisco Meraki

Cisco Meraki documentation provide detail changelog and version details of each change they have made to their API interface over the time. At the time of this research a newer major version of the API was announced and there was noticeable documentation online about retirement date of previous version as shown in the snapshot below

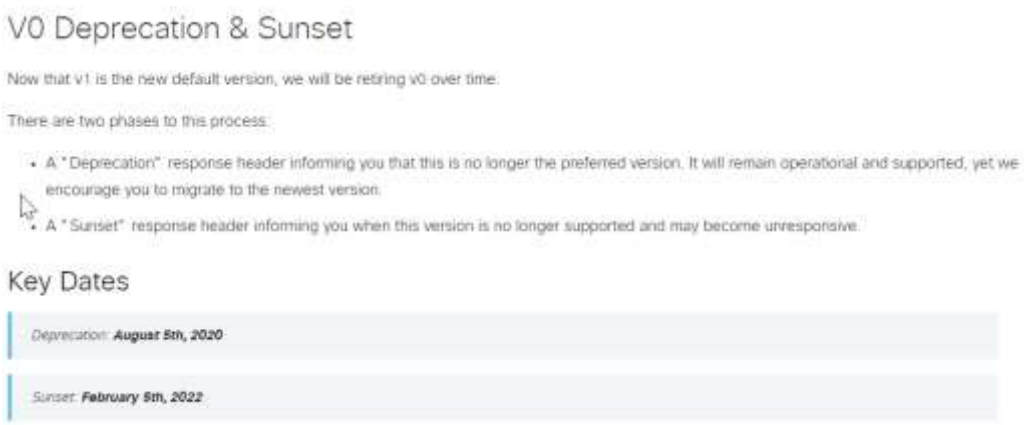


Figure 12: API versioning example from Cisco Meraki online documentation

Apart from major versioning detail changelog was maintained and was publicly available on Cisco Meraki website on what changes has been done every month. Changes include new API calls or any changes to information required or sent in respond of a call.

Figure 13: Changelog example from Cisco Meraki documentation

Even slightest change to API was documented and was publicly available to help in integration of the API interface

Figure 14: API enhancement snapshot from Cisco Meraki Documentation

V2A3 Integration – VMware Velocloud

VMware Velocloud documentation did not have any details related to API versioning.

Usage and Telemetry

Vendor documentation and online portal will be used to check the usage and telemetry stats against the API requests.

V1A4 Usage and Telemetry – Cisco Meraki

Cisco Meraki portal only provide information about when was the API call last used. There might be API usage and telemetry data collected by the vendor, but it was not exposed to end user.

V2A4 Usage and Telemetry – VMware Velocloud

VMware VeloCloud online portal and documentation did not provide any usage and telemetry stats. There might be API usage and telemetry data collected by the vendor, but it was not exposed to end user.

Developer Experience

No data is collected to gauge the developer experience. This rating is based upon the overall experience of the interaction of the API interface. The guideline for this aspect is rate based on ease of adopting specific vendor API.

V1A5 Developer experience – Cisco Meraki

For this research, adopting and using Cisco Meraki APIs were comparatively easier due to up-to-date documentation and usage examples online. Apart from clear documentation a postman collection of API call was provided by the vendor to easily import the collection inside the Postman.

V2A5 Developer experience – VMware Velocloud

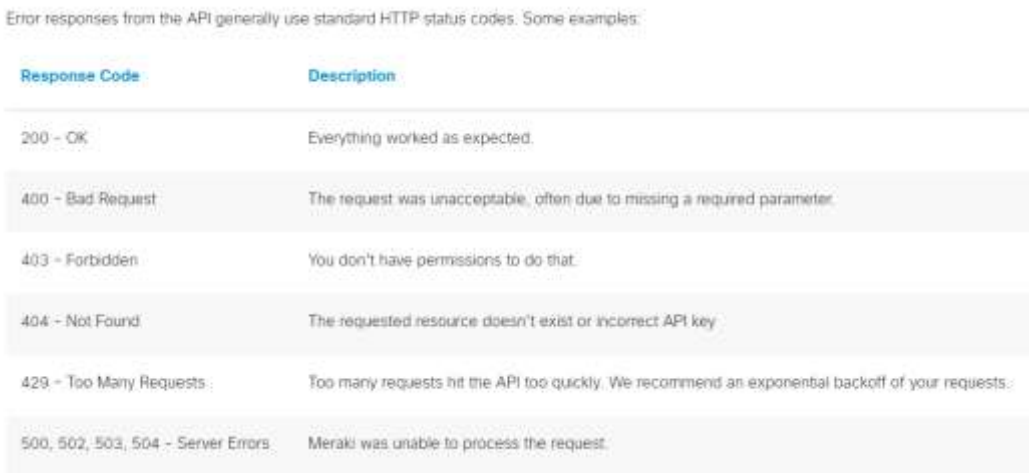
For this research, adopting and using VMware VeloCloud APIs were comparatively difficult due documentation not being that clear and a smaller number of examples available online. A postman collection of API call was provided by the vendor to easily import the collection inside the Postman

Error Handling

Postman was used to check the error response code and to check if standard HTTP codes were following in Error responses and if how descriptive error messages in the code were for different reasons when request fail for different reasons.

V1A6 Error Handling – Cisco Meraki

Cisco Meraki was following standard error codes



Error responses from the API generally use standard HTTP status codes. Some examples:

Response Code	Description
200 - OK	Everything worked as expected.
400 - Bad Request	The request was unacceptable, often due to missing a required parameter.
403 - Forbidden	You don't have permissions to do that.
404 - Not Found	The requested resource doesn't exist or incorrect API key.
429 - Too Many Requests	Too many requests hit the API too quickly. We recommend an exponential backoff of your requests.
500, 502, 503, 504 - Server Errors	Meraki was unable to process the request.

Figure 15: Response Codes example from Cisco Meraki documentation

If the response code was not Specific enough to determine the cause the response body contain the detail description about the reason request failed

```
1. {
2.     "errors": [
3.        "IP Address does not exist on this network"
4.     ]
5. }
```

V2A6 Error Handling – VMware Velocloud

VMware Velocloud documentation didn't provide that many details of the error handling and what each error means. Unsuccessful attempts did return with standard http response code for error and body of the message contain description of the error and error code which might be useful for internal documentation of the VMware.

```
1. HTTP/1.1 400
2. status: 400
3. Server: nginx
4. Date: Wed, 19 Aug 2020 12:18:53 GMT
5. Content-Type: application/json
6. Content-Length: 95
7. Connection: keep-alive
8. X-Powered-By: Express
9. Cache-Control: no-cache,no-store,must-revalidate
10.  Pragma: no-cache
11.  Expires: 0
12.  {"error":{"code":-32603,"message":"unable to find enterprise
   for operator enterprise context"}}
```

Some observations in the above code:

- Line Number 2: Correct Error code 400 is returned

- Line Number 12: Response body contain Error code and detail message why this request was failed.

With another example where request failed but wrong response code was sent in response. Although response body contain the detail and reason for the failed request,

```
1. HTTP/1.1 200
2. status: 200
3. Server: nginx
4. Date: Wed, 19 Aug 2020 12:25:44 GMT
5. Content-Type: application/json
6. Content-Length: 123
7. Connection: keep-alive
8. X-Powered-By: Express
9. Cache-Control: no-cache,no-store,must-revalidate
10. Pragma: no-cache
11. Expires: 0
12. Strict-Transport-Security: max-age=31536000; includeSubdomains;
13. X-Frame-Options: SAMEORIGIN
14. {"id":1597839944519,"jsonrpc":"2.0","error":{"code":-
    32000,"message":"tokenError [credential for authentication
    missing]"}}
```

Some observations in the above code:

- Line Number 2: Wrong Error code 200 is returned
- Line Number 12: Response body contain Error code and detail message why this request was failed.

Above two examples show that error handling was not consistent for VeloCloud API and also the lack of documentation make their API integration much more difficult.

Community

Community aspect has been added by this research to API testing framework. Community was gauge by number of active threads on the community pages and overall activeness by gauging the number of activities in last month on the community portal.

V1A7 Community – Cisco Meraki

Cisco Meraki community page was vibrant and full of actively involved members. There were 16 new posts within last week with some posts with more than 10 replies. There was clear focus by the vendor on community development where top authors and solution authors were rewarded with the badges and incentives.

Figure 16: Community reward and encouragement example from Cisco Meraki Documentation

Label TAGs were used to differentiate differet type of topics to filter the post with respect to area of interest

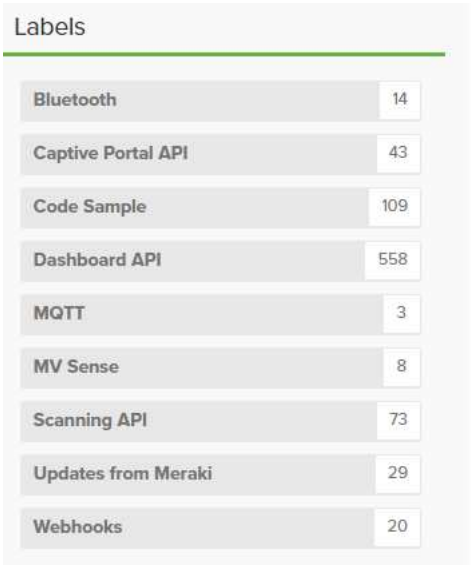


Figure 17: Use of Labels example from Cisco Meraki community page

V2A7 Community – VMware Velocloud

VMware Velocloud doesn't provide a separate community for API but there is a community page for overall VMware VeloCloud SD-WAN solution which can be used to ask API related queries. There is close to no activity on the community page as there was only one new topic initiated in last one week.

Documentation

Documentation is the 2nd aspect that has been added by this research to API testing framework. Documentation is gauge by accuracy of the documentation. Documentation accuracy and completeness was gauge based on usefulness of documentation to make API calls for testing of this framework.

V1A8 Documentation – Cisco Meraki

Cisco Meraki provide complete and accurate documentation of API interface. Detail documentation was available for each aspect of the API including how to use it via Postman, versioning, error handling and detail examples. The documentation was followed by strong community and it was evident from the community activities that vendor has strong focus on programmability and API usage of their solution.

V2A8 Documentation – VMware Velocloud

VMware documentation was lackluster. VMware do have online documentation using swagger and some examples available online on how to use APIs. There was guide provide to use the API, but documentation was limited and not concise enough.

3.7 Conclusion

In this chapter we discussed the research design and testing methodology and how this testing methodology can be used for evaluating any SD-WAN vendor. We covered the reason for vendors selection for this research. The chapter also cover the data acquisition methods and concept around the data formats. Lastly collected data was presented for all eight aspects of the testing framework for both selected vendors.

4.0 Results

4.1 Introduction

This chapter reports the finding and results obtained by the research by applying the testing framework on two selected SD-WAN vendors. This chapter discuss and analysis the collected data, observations made during data collection and results of applied framework to the collected data. The chapter list down the result of each aspect and provide both vendor score for each aspect of testing framework along with the reasoning of assigned score and calculate weighted score by applying weightage of each aspect to the score given to each vendor. The chapter conclude by providing detail analysis of the results

4.2 Analysis

In this section of the chapter, we analysis the collected data and apply the testing framework to calculate how each vendor scored for each aspect of the framework. We will assign a weightage (W) for each aspect and will list down the reasoning for assigned weightage. A researcher may use different value of W based on how each aspect is critical for his requirements and evaluation both same W value needed to be applied for all the vendors that are being evaluated.

The value can be any value for each aspect between 0, 0.1, 0.2, up to 1. Where 1 being highest where that aspect is most critical for the research while 0 being lowest where that score for that aspect will have no impact in vendor evaluation.

Each vendor will be assigned a score S between 1 to 10 and based on how that vendor performed in that aspect. This section will also discuss the assigned score to both vendors and rational behind the given value of S.

The testing framework consists of eight aspects. These aspects are listed below followed by the description of each aspect.

- Security
- Performance
- Integration
- Usage & Telemetry
- Developer Experience
- Error Handling
- Community
- Documentation

Security

The security is most important aspect of the architecture as for SD-WAN most of the controller and orchestrator those have the API interface are located on the internet subjects to all kind

of exploits without any firewall or filters in front of the API interface. The key feature security cover is authentication and authorization of the API requests.

Apart from Authentication and authorization security also cover protection against misuse by legitimate user which is done by rate-limiting the requests to a specific number in a given time. Rate limiting works two-fold one it secures the API interface from misuse and secondly it protects the server resources hogging that might require to respond to many other simultaneous API calls. There are multiple ways of rate limiting and different vendor may adopt none, one or more ways for rate limiting the API calls. Some ways to implement rate limiting are given below:

- Rate limiting the overall number of calls a server will respond in a given time regardless of source of the API calls.
- Rate limiting the number of calls a user can make in given time.
- Rate limiting the volume of data a user can request in a given time.
- Rate limiting the number of calls from a specific source.

The most common way to rate limit is to limit the number of calls in given time for example a single token assigned to a user can only make five calls in a second.

Another aspect security cover is the logging of the API calls, which can be used for troubleshooting and forensic analysis of the events specifically for the case of restful API's POST, PUT or UPDATE events where changes are made to the resources. Logging can be as detail as logging each and every API call with timestamp and can be minimal of when token was created and last used.

Due to all these different aspects security cover the Weightage of 1 is given to security which is highest weightage an aspect can get in this framework

Weightage for security $W_1 = 1$

Security is the only aspect of the testing framework which has been assigned highest weightage of 1

Security Score for Cisco Meraki

Collected data showed that Cisco Meraki controller use an API key for authorization. Each request needs to have this key to get authorized on the server. The key is passed to server in the header of the request. Since these API keys can be used to gain access to all kind of data on the controller these API keys needed to be treated like password. Like any other password its user responsibility to keep it secure and to follow the best practices like periodically regenerate the new API key.

On Cisco Meraki portal you can only see the key when its generated. If key has been compromised it can be revoked with click of a button and a new key can be generated.

Figure 18: API key generate and revoke example from Cisco Meraki dashboard

As shown above the key value cannot be seen even by the user who own and generated the key. Basically, the key generated on Cisco Meraki Controller cannot be revealed by any mechanism available to user from Meraki controller. From user perspective if a key has been lost or compromised then that key will required be to revoke from Meraki user portal a new key will be generated by user for authentication.

Collected data also showed that Cisco Meraki portal provide rate limiting documentation where 5 API calls can be made for one organization. Additionally, a burst of 5 additional calls are allowed for first second that mean in first two seconds the Meraki server will allow up to 15 calls against an organization. If a request is failed due to rate-limiting a failure message with error code of 429 is send to the user which can come handy where user can write down code to add delay to the subsequential request calls if they receive error code 429 in response.

Collected data showed that the Meraki portal show the last usage of the API key. There were no stats on usage of the key and calls log for each call request.

All the above observation conclude that Cisco Meraki has secure their API interface on multiple folds. It authorises each request, there is rate-limiting implemented for securing the server against misuse and from API calls impacting performance of the server and lastly it showed that the API key last usage is available on their portal. Cisco Meraki can improve further in logging where log of each API calls specifically where changes has been made using API calls should be logged for auditing and forensic. Based on all these security measure this research assigns a score of 8 out of 10 to Cisco Meraki.

Cisco Meraki security score $S_{11} = 8$

Cisco Meraki weighted security score = $W_1 * S_{11} = 1 * 8 = 8$

Security Score for VMware VeloCloud

Collected data showed that VMware required to send a key value pair of authorization and token to the server for authentication. The implementation is bit wired where token keyword is required in the value of the key as shown below:

Key	Value
Authorization	Token XXXXX

Table 3: Key Value Pair adoption of VMware VeloCloud

Where XXXX is the API key token generated on Velocloud portal. From authorization perspective it served the purpose but is not standard design for key value pair. This could have been easily improved by naming the key as Authorization-Token and putting token value XXXX in the field.

The collected data also showed that VeloCloud also authenticate based on username and password where response is a cookie that needed to be passed to the server for subsequent requests. The issue with this cookie-based authorization is that such authorization is stateful in nature that is server need to keep track of the cookies for each request to keep session information for authorizing subsequent request. As these cookies are temporary session management needed to be implemented on the server side. This stateful nature of the authorization where server need to keep the session details is problematic from scaling point of view and it doesn't fit well from REST paradigm. The SD-WAN vendor APIs are designed to scale as these servers are multi-tenant and multiple users sometime in thousands can be sending simultaneous request to the server which can cause performance issues on the server. Cookie based authorization is also vulnerable to CSRF/XSRF (Cross Site Result forgery) as attacker can take advantage of previously authenticated session and replay it for its own malicious purpose[19].

Collected data also showed that VeloCloud API didn't had any document rate limiting policy at the time of this research. No rate limiting make the server vulnerable to API calls and user abusing legitimate and illegitimate use of the server. The server can be overwhelmed just to respond to the API calls and a simple script of generating thousands of calls cause denial of service to other legitimate users.

Collected data showed that there is no log available to users about usage of the API calls. The information that is available against a generate token is creation date.

Token do come with pre-set expiration date which secure these tokens from being active forever but lack of any visibility on use of token is something which needed to be improved on.

All the above observation conclude that VMware Velocloud need to improve their API interface security. The cookie-based authentication, no rate-limiting and no usage logging visible to end user make VMware Velocloud security very weak, based on all these security issues this research assign a score of 4 out of 10 to VMware VeloCloud.

VMware VeloCloud security score $S_{21} = 4$

VMware VeloCloud weighted security score = $W_1 * S_{21} = 1 * 4 = 4$

Performance

Performance covers the responsiveness of the API interface. The testing of performance is usually done using stress testing of the API using multiple calls. The performance aspect of the

API also covers the scalability and availability of the API calls. Scalability is how many simultaneous calls an API interface can handle while availability is basically uptime of the API interface to serve the API calls.

Performance of any API call can be improved using caching and pagination mechanisms. Caching is usually used where same information request is anticipated from multiple calls in this way response to first call is cached in the server and subsequent request get the copy of that cache. Cache add performance as well as help to improve the scalability of the interface.

Another performance improvement mechanism used by the vendor is pagination where instead of trying to all the matched information by request call server send handful of record to the client and expect from client to ask for remaining portion of information in subsequent requests. Pagination help server to save the memory where huge amount of data is not required to be calculated for each request but only a selective chunk of it.

The performance aspect is important mainly for end user and for vendor itself but do not play important role from integration point of view therefore this research give a weightage of 0.1 to performance

Weightage for performance $W_2 = 0.1$

Weightage 0.1 is the lowest weightage given to any aspect of this framework for this testing.

Performance Score for Cisco Meraki

Collected data showed that Cisco Meraki API was responsive during the testing. The metric use to test the responsiveness of the API call was time it took to get the response from server after an API call.

Due to low weightage of performance aspect no stress testing of the API interface was done for this research and the metric of response time was dependent upon the distance of the client from the server that responding to the API call.

Collected data showed that there was no caching mechanism was documented by Cisco Meraki API. Meraki API do provide pagination for GET requests where subset of the information is requested in the first call and rest of the information in subsequent calls.

All the above observation conclude that Cisco Meraki was responsive and have implemented pagination mechanism to improve the performance of the API Cisco Meraki can improve further by adding caching mechanism to the similar requests. Based on all these observations this research assigns a score of 7 out of 10 to Cisco Meraki

Cisco Meraki performance score $S_{21} = 7$

Cisco Meraki weighted performance score = $W_2 * S_{21} = 0.1 * 7 = 0.7$

Performance Score for VMware VeloCloud

Collected data showed that VMware VeloCloud API was responsive during the testing. The metric use to test the responsiveness of the API call was time it took to get the response from server after an API call.

No stress testing was performed on VMware VeloCloud API interface and the metric of response time was dependent upon the distance of the client from the server that responding to the API call.

Collected data showed that there was no documented cacheing mechanism listed on publicly available documentation of VMware VeloCloud API.

Collected data showed that there was no pagination mechanism was documented by VMware VeloCloud API.

All the above observation conclude that VeloCloud VMware was responsive but have implemented neither caching nor pagination mechanism to improve the performance of the API requests and performance of the API can improve further by adding cacheing and pagination mechanism to the API architecture. Based on all these observations this research assigns a score of 4 out of 10 to VMware VeloCloud.

VMware VeloCloud performance score $S_{22} = 4$

VMware VeloCloud weighted performance score = $W_2 * S_{22} = 0.1 * 4 = 0.4$

Integration

Integration covers the API changelog and version details. Recent change longs on API changes reflect two characteristics of the vendor; one that the API vendor is spending time and effort to improve the API over time and secondly that the API version information is available to the users and users will not hit by sudden change in the API which may impact the usage and API interface response.

Not only change logs and versioning is required but also there needed to be enough overlap between two major versions so that users get ample amount of time to do the changes on their API code to update the calls for the new version before the older version of the API expires. There are multiple ways to implement versioning and most common way adopted by most of vendors is adding the version number in the baseURL of the API interface so that one its clearly visible and second easily identifiable when developer or user working with multiple versions of the code during migration phase for developer where some code changes is required in previous version to fix some issue within backward compatibility time announced by the vendor. The backward compatibility time differ vendor to vendor but at minimum six months of backward compatibility is retained which given ample amount of time for requester to update and migrate to newer version of the code.

The integration aspect is key aspect given that the Telcos are usually slow to react to changes and take time to develop and update their fulfil, assure and billing stack even some time they are dependent upon their other vendors when they use APIs to connect two different solutions therefore this research gives a weightage of 0.4 to integration

Weightage for integration $W_3 = 0.4$

Integration Score for Cisco Meraki

Collected data showed that Cisco Meraki had detail versioning for any major change to the API version publicly available on their website. The Vendor did not depreciate the old version straight away and date of depreciation was publicly available and there was a period of two years between announcement of new major version and deprecation date of the older version of the API.

The Collected data also showed that detail changelogs were maintained by the vendor for each small change on their API interface. All the new API calls were documented with examples on how to use them and if there were any changes to existing API request like additional of any new key value pair in returned data it was properly documented with difference shown in previous and new response.

All the above observation conclude that Cisco Meraki hold highest standard on the integration aspect of the API interface. Not only there is major versioning with clear announcement of depreciation data and minor changes were documented in detail for users with appropriate examples. Based on all these observations this research assigned a score of 9 out of 10 to Cisco Meraki for integration aspect.

Cisco Meraki integration score $S_{31} = 9$

Cisco Meraki weighted integration score = $W_3 * S_{31} = 0.4 * 9 = 3.6$

Integration Score for VMware VeloCloud

Collected data showed that VMware VeloCloud did not published any public details for its API versioning. This possibly because VMware VeloCloud is still on its first major version, but this was not indicated anywhere in the documentation either and there was no public roadmap for any future version to be released.

The VMware documentation did not show any changelog that is being maintained by the vendor. If the changes were done by vendor to its API interface there was no record of such changes publicly available. This can possible be because there were no recent changes made by the vendor but there was not any historical changelog available in VMware documentation.

All the above observation conclude that VeloCloud VMware is maintaining very low standard for its integration aspect. Having no major or minor versioning available and having no changelog publicly maintained for API interface show weak integration and can cause many issues if API interface is used by Telcos for integration. Based on all these observations this research assigned a score of 2 out of 10 to VMware VeloCloud on integration aspect.

VMware VeloCloud integration score $S_{32} = 2$

VMware VeloCloud weighted integration score = $W_3 * S_{32} = 0.4 * 2 = 0.8$

Usage & Telemetry

Usage and telemetry cover the statistics of overall API usage and details statistics of the API calls. These statistics can be how many calls are being made on a given period. What are the top requests that are being made and what the success rate for these calls? More statistics can be performance related that is how much data is being returned on average for each call and how much time server is taking to response on average for each call.

API usage stats can be useful to track the consumption of the API and can also use to gauge the performance of the API interface.

The usage & telemetry provide useful data for both developer and the integration and this research give weightage of 3 for usage and telemetry

Weightage for usage and telemetry $W_4 = 0.3$

Usage & telemetry score for Cisco Meraki

Collected data showed that the usage and telemetry stats for the API are not available to the users.

The only stats about the API usage that can be found was based on the API key that is when the key was created and when it was last used.

All the above observation conclude that Cisco Meraki API has lot to catchup in the usage and telemetry side of the APIs. This research assigned a low score of 3 out of 10 to Cisco Meraki for usage and telemetry aspect.

Cisco Meraki usage and telemetry score $S_{41} = 3$

Cisco Meraki Weighted usage and telemetry score = $W_4 * S_{41} = 0.3 * 3 = 0.9$

Usage & telemetry score for VMware VeloCloud

Collected data showed that the usage and telemetry stats for the API interface are not available to the users.

The only stats about the API usage that can be found was based on the token that is when the token was created. For the token VMware did not provided any state on when was the last this token was used.

All the above observation conclude that VMware VeloCloud API has lot to catchup in the usage and telemetry side of the APIs. This research assigned a low score of 2 out of 10 to VMware VeloCloud for usage and telemetry aspect.

VMware VeloCloud usage and telemetry score $S_{41} = 2$

VeloCloud Weighted usage and telemetry score = $W_4 * S_{41} = 0.3 * 2 = 0.6$

Developer Experience

Developer experience is based on how easy it was developer to adopt vendor provider APIs. It covers lot of aspects of the APIs but main aspect where APIs shine are standardisation. The more an API follow the standard development guidelines the easier it will be for developer to adopt the API and use it for integration of two systems. Another aspect of the developer experience is use of the API as the main purpose of the API is machine to machine communication and for developer it will be easier for two machines to communicate if its following standard and less tweaking is required from the developer to make them talk to each other.

The Developer experience is important aspect, but as no actual integration was done for the research with telco fulfil, assure and billing stack this research gives low weightage of 2 for developer experience

Weightage for developer experience $W_5 = 0.2$

Developer experience score for Cisco Meraki

Researcher experienced no issue in adopting Meraki API interface. As Postman was used as client, Cisco Meraki made API calls collection library specifically for Postman usage online which was downloaded and used for this testing. Developer noted that Cisco Meraki was following the standard guidelines for restful API calls. API calls were responding as documented.

All the above observation conclude that developer had good experience with Cisco Meraki APIs during the testing. Cisco Meraki is following standard guidelines, which has made API interface more useable for developers. This research assigned a score of 9 out of 10 to Cisco Meraki for Developer experience aspect.

Cisco Meraki developer experience score $S_{51} = 9$

Cisco Meraki Weighted developer experience score = $W_5 * S_{51} = 0.2 * 9 = 1.8$

Developer experience score for VMware VeloCloud

Like Cisco Meraki, VMware Velocloud had their API calls available online. The VMware provided the documentation in the format of a swagger file which was easily imported to Postman client to make calls for API.

The API usage was not smooth sailing for the case of VMware mainly due to not following the standard guidelines. The major issue with the VMware was that each call was POST call regardless if it was even to get an information from the VMware API interface. This cause of lot of confusion and consumed time in understanding how these calls are working.

Setting API authorization also required multiple attempts because of VeloCloud using additional keyword Token before the token value in the value field of the header.

All the above observation conclude that VeloCloud VMware is difficult to adopt, and additional time and effort is required to understand and consume non-standard API calls of VMware VeloCloud. Based on all these observations this research assigned a score of 3 out of 10 to VMware VeloCloud on developer experience aspect.

VMware VeloCloud developer experience score $S_{52} = 3$

VeloCloud Weighted developer experience score = $W_5 * S_{52} = 0.2 * 3 = 0.6$

Error Handling

Error handling is one of the key aspects testing framework. Error handling help in integration two folds. One it helps integrator to understand if request is unsuccessful why its failed and secondly it allow developer to write error handling code where integrator can make different call.

Error handling is where some vendors do not pay enough attention and badly return API can return same error code for multiple reason without explaining the reason for why API call fail whereas a properly developed API will return different error code as well as error message for different reasons when API calls is failed to let integrator know what needed to be changed in the request.

A proper error code along with error message will not only help the integrator to understand the code but will also allow to further extend his API calls by writing error handling calls. For example, if error message contains that the VLAN number is missing in the request, integrator can write a code that in response of this error code make two additional calls one to get the VLAN number and secondly use the return VLAN number to again retry the previous attempt with the VLAN number.

The error handling is one of the most important factors in long term usability of the APIs as it reduces time many folds when things not going as per plan. This research gives high weightage of 0.9 for error handling.

Weightage for error handling $W_6 = 0.9$

Error handling score for Cisco Meraki

Data collected on error handling of Cisco Meraki showed that the Cisco was using standard HTTP status codes for both successful and failed attempts. For most of the instances response status code was good enough for understanding the root cause for an API call to failed. Apart from response code the body of the response had a detail description and error message about why request failed.

This standard error status code made it easier for integrator to change their code to handle the error. For example, Cisco Meraki always used the status code of 429 when a request fails due to rate-limiting applied on API interface. An integrator can easily add delay to its next request to overcome the rate-limiting applied by Cisco whenever a response message of 429 is received against a call.

All the above observation conclude that Cisco Meraki has developed its API with proper error responses and all the response messages has been publicly documented for integrator to use them to troubleshoot and to develop error handling codes to alter their next request. Based on all these error handling provided by Cisco Meraki this research assigns a high score of 9 out of 10 to Cisco Meraki

Cisco Meraki error handling score $S_{61} = 9$

Cisco Meraki weighted error handling score = $W_6 * S_{61} = 0.9 * 9 = 8.1$

Error handling score for VMware VeloCloud

Data collected on error handling by VMware showed that the VMware respond to errors with standard HTML status codes but there is no documentation publicly available on what error code is sent when request failed for specific reason.

Below are two examples of failed attempt and some observations on the attempts

```
1. POST /portal/rest/enterprise/getEnterprise
2. Content-Type: application/json
3. Authorization: Token XXXXX
4. User-Agent: PostmanRuntime/7.15.0
5. Accept: */*
6. Cache-Control: no-cache
7. Postman-Token: 8020ebcf-0a4f-4cd5-985f-559dcc4eb0e5
8. Host: vco303-syd1.velocloud.net
9. accept-encoding: gzip, deflate
10.  content-length: 40
11.  Connection: keep-alive
12.  { "id": 3, "enterpriseId": 773 }
13.  HTTP/1.1 400
14.  status: 400
15.  Server: nginx
16.  Date: Sat, 22 Aug 2020 06:26:56 GMT
17.  Content-Type: application/json
18.  Content-Length: 95
19.  Connection: keep-alive
20.  X-Powered-By: Express
21.  Cache-Control: no-cache,no-store,must-revalidate
22.  Pragma: no-cache
23.  Expires: 0
24.  {"error":{"code":-32603,"message":"unable to find enterprise
   for operator enterprise context"}}
```

Some observations from above code of first example:

- Line Number 14: Error code 400 is returned which showed that the attempt failed
- Line Number 24: Contain the error code 32603 which some error code internal to VMware VeloCloud
- Line Number 24: Contain the error message that it is not able to find the enterprise

```
1. POST /portal/rest/enterprise/getEnterpriseAddresses
2. Content-Type: application/json
3. User-Agent: PostmanRuntime/7.15.0
4. Accept: */*
5. Cache-Control: no-cache
6. Postman-Token: 0f70de72-6ba6-4120-86a1-ef7e6d059bfd
7. Host: vco303-syd1.velocloud.net
8. accept-encoding: gzip, deflate
9. content-length: 25
10.  Connection: keep-alive
11.  { "enterpriseId": 3 }
12.  HTTP/1.1 200
13.  status: 200
14.  Server: nginx
15.  Date: Sat, 22 Aug 2020 06:27:48 GMT
16.  Content-Type: application/json
17.  Content-Length: 123
18.  Connection: keep-alive
19.  X-Powered-By: Express
20.  Cache-Control: no-cache,no-store,must-revalidate
21.  Pragma: no-cache
```

```
22. Expires: 0
23. Strict-Transport-Security: max-age=31536000; includeSubdomains;
24. X-Frame-Options: SAMEORIGIN
25. {"id":1598077668604,"jsonrpc":"2.0","error":{"code":-
    32000,"message":"tokenError [credential for authentication
    missing]"}}
```

Some observations from the above code of second example:

- Line Number 13: Status code 200 is returned which indicate the attempt was successful, but the message body indicate it was a failed attempt.
- Line Number 25: Contain the error code 32000 which is some error code internal to VMware VeloCloud
- Line Number 24: Contain the error message that the authentication token is missing.

As shown using above two examples that although different error codes are returned in the body of the respond the details of error code are missing from the vendor documentation. Similarly, no documentation on how to overcome this error code or what action is required to resolve this error is publicly available. The second example also highlight that the VMware VeloCloud API return wrong status code where error code 200 was returned which generally mean that the attempt was successful, but examination of returned body showed that the request failed due to authentication token was missing in the request.

The respond code does contain the error messages and an error handling code can be written to overcome these errors, but VMware implementation made it difficult by making error code part of body. This difficulty is increased due to no publicly available documentation or examples on writing error handling code.

All the above observation conclude that VeloCloud VMware leave a lot to ask for error handling and integrator will have difficult time to troubleshoot and writing error handling code. Based on all these observations this research assigned a score of 4 out of 10 to VMware VeloCloud on developer experience aspect.

VMware VeloCloud error handling score $S_{62} = 4$

VMware VeloCloud weighted error handling score = $W_6 * S_{62} = 0.9 * 4 = 3.6$

Community

Community aspect has been added by this research to API testing framework. An active and vibrant community help in many ways. One it shows that API interface is being adopted by multiple users, secondly provide a platform for API users to communicate and exchange ideas. Community portal is also good to get customer feedback on what is missing in the API interface and how vendor can further improve the API. The community portal usually serves as collective feature request where community member can vote collectively to shortlist the features which developer can prioritise its development for users.

These active community members usually also best candidate to roll out beta features in the testing where code is not finalized. The vendor can release the features to controlled group who are encouraged by the vendors to do beta testing from users and integrator perspective.

A vibrant and active community help both integrator and API vendors as both can get benefits from it. From integrator perspective community portal serve as Wiki pages for looking for solutions to see if any user faced same issue and how that issue was resolved. This research give weightage of 0.5 to community aspect of the framework.

Weightage for community handling $W_7 = 0.5$

Community score for Cisco Meraki

Collected data showed that Cisco as a vendor was investing heavily to develop an active and vibrant community and community page was lively at the time of this research. It was evident that community is active as there were 16 new posts on the vendor community portal and these posts were from end users not from vendor itself and some of posts had more than 10 communication and replies on the portal.

Cisco was awarding top authors by badges and other incentives. The number of top user actives and solution provided by users that accepted by the requester were also highlighted.

Cisco was also labelling different portal pages based on area of interests so that community member can filter out and can look for community pages on specific topic.

All the above observation conclude that Cisco Meraki has developed an active and vibrant community and is investing in keeping this community involved. Based on these observations this research assigns a high score of 8 out of 10 to Cisco Meraki on community aspect of the testing framework

Cisco Meraki community score $S_{71} = 8$

Cisco Meraki weighted community score = $W_7 * S_{71} = 0.5 * 8 = 4.0$

Community score for VMware VeloCloud

Data collected on community showed that VMware has not invested much in community portal and there was no encouragement by vendor to attract users to its community portal. To start with there was no separate community portal for API usage. There was a community portal for overall usage of VMware VeloCloud SD-WAN solution. Even there was very low activity on overall solution community page.

It was observed that only one new post was listed in last one week on the VMware Community page where this community page is not specific to API but was created for whole SD-WAN product of the vendor.

There was no incentive to users to join or participate on the community page

All the above observation conclude that VeloCloud VMware need invest heavily to develop an active and vibrant community. Based on all these observations this research assigned a score of 2 out of 10 to VMware VeloCloud on community aspect.

VMware VeloCloud community score $S_{72} = 2$

VMware VeloCloud weighted community score = $W_7 * S_{72} = 0.5 * 2 = 1.0$

Documentation

Documentation is 2nd aspect has been added by this research to API testing framework. Documentation is gauge by completeness and as well as accuracy of the documentation. A documentation improves the integrator experience and make it easy for them to integrate and use API. API documentation touch all other aspects of API testing framework as it serves as instructions on how to use the APIs.

The good API documentation on one side it helps user to easily consume API interface and on the other hand it saves vendor time and cost on not to answer the integrator queries where a good documentation could have served the purpose.

For the integrator learning a new API take time and learning curve is steep. A good documentation decrease that learning curve steep.

A well written API code with bad documentation is worst then a badly written API code with good documentation. Due to high importance of documentation, this research give weightage of 0.9 to documentation aspect of the framework.

Weightage for documentation handling $W_9 = 0.9$

Weightage of 0.9 is 2nd highest weightage in this research.

Documentation score for Cisco Meraki

Data collected about documentation of Cisco Meraki showed that Cisco Meraki has done provided a comprehensive and accurate documentation.

Detail documentation was available publicly online and there were many tutorial and example codes and videos available on how to start with the adoption and learning of the APIs. The Cisco Meraki provide downloadable Postman collection file which was used for this testing.

Documentation was comprehensive for each aspect including versioning, error handling, detail examples and changelogs.

All the above observation conclude that Cisco Meraki has done an admirable job in providing complete and accurate documentation of the API interface. Based on these observations this research assigns a high score of 9 out of 10 to Cisco Meraki on documentation aspect of the testing framework

Cisco Meraki documentation Score $S_{81} = 9$

Cisco Meraki Weighted documentation score = $W_8 * S_{81} = 0.9 * 9 = 8.1$

Documentation score for VMware VeloCloud

Data collected showed that VMware documentation was not complete and lacked accuracy.

VMware documentation did provide a swagger file which helped in this testing as that swagger was imported into Postman client as collection but documentation on how to use this collection and what to expect in response was missing.

All the above observation conclude that VeloCloud VMware need to publish a more detail and complete version of its documentation publicly to make adoption of its API interface easier for the integrator. Based on all these observations this research assigned a score of 3 out of 10 to VMware VeloCloud on documentation aspect.

VMware VeloCloud documentation Score $S_{82} = 3$

VMware VeloCloud Weighted documentation Score = $W_8 * S_{82} = 0.9 * 3 = 2.7$

4.3 Results Analysis

Aspects	Weightage (0 – 1)	Cisco Meraki score (1 – 10)	Cisco Meraki weighted score	VMware VeloCloud score (1 – 10)	VMware VeloCloud weighted score
A1 – Security	W_1	S_{11}	W1*S11	S_{21}	W1*S21
A2 – Performance	W2	S_{12}	W2*S12	S_{22}	W2*S22
A3 – Integration	W3	S_{13}	W3*S13	S_{23}	W3*S23
A4 – Usage & Telemetry	W4	S_{14}	W4*S14	S_{24}	W4*S24
A5 – Developer Experience	W5	S_{15}	W5*S15	S_{25}	W5*S25
A6 – Error Handling	W6	S_{16}	W6*S16	S_{26}	W6*S26
A7 – Community	W7	S_{17}	W7*S17	S_{27}	W7*S27
A8 – Documentation	W8	S_{18}	W8*S18	S_{28}	W8*S28
Total			SV_1		SV_2

Table 4: Framework aspects and vendor Score and weightage

Aspects	Weightage (0 – 1)	Cisco Meraki score (1 – 10)	Cisco Meraki weighted score	VMware VeloCloud score (1 – 10)	VMware VeloCloud weighted score
A1 – Security	1	8	8.0	4	4.0
A2 – Performance	0.1	7	0.7	4	0.4
A3 – Integration	0.4	9	3.6	2	0.8
A4 – Usage & Telemetry	0.3	3	0.9	2	0.6
A5 – Developer Experience	0.2	9	1.8	3	0.2
A6 – Error Handling	0.9	9	8.1	4	3.6
A7 – Community	0.5	8	4.0	2	1.0
A8 – Documentation	0.9	9	8.1	3	2.7
Total			35.3		13.3

Table 5: Assigned weightage, vendor score and weighted score of vendors

Cisco Meraki API scored a very high weighted score of 35.3 from possible total of 43

VMware VeloCloud API scored a lackluster weighted score of 13.3 out of 43

The research showed that although both are leading SD-WAN vendors, but on API front Cisco come out as clear winner. Not only its overall score was high but, in each aspect, Cisco Meraki scored higher compare to VMware VeloCloud. The only aspect where Cisco Meraki need

significant improvement is usage and telemetry, even for that aspect, Cisco Meraki scored higher score compare to VMware VeloCloud.

4.4 Conclusion

The result showed that the API testing framework was able to differentiate between a good implementation of API and comparatively a bad implementation of the API. The framework can be adopted for testing of different vendor and weightage factor provide flexibility to change framework with respect to researcher requirements.

5.0 Discussion

This chapter discusses the research findings and if the questions have been answered by this research. The discussion is based on results in chapter four. Furthermore this chapter discuss if other architects can adopt this framework to compare their short-listed SD-WAN vendors. Chapter discuss each aspect and their score and then in the end discuss the overall score of the vendors.

5.1 Introduction

The results and findings for both the vendors using the API testing framework were presented in descriptive and tabular format in previous chapter. The target of this chapter is to analyze and discuss the results of the SD-WAN API framework applied in previous chapter and use the finding and results to answer the main research questions.

5.2 Answers to research questions

First Question

"Could the existing framework as defined in API architecture and development (Vijayakumar, 2018) can be directly applied to test the usefulness of SD-WAN APIs for integration with service provider existing stack?"

First Question Answer

No, the existing framework cannot be directly applied to test the usefulness of SD-WAN APIs for integration with service provider existing stack. The existing framework were designed for cloud service provider and were covering the aspects which were not related to SD-WAN vendors for example API gateways.

The existing framework did serve as foundation of the new framework presented in this research. The research studied the existing framework and remove the aspects from the framework which are not related to SD-WAN vendor and added the aspects which are crucial for SD-WAN vendor's provided API evaluation.

The research concluded that the modification to existing framework like adding aspects like community and documentation are crucial for testing SD-WAN vendor APIs specially if the testing is to gauge the usefulness of API for integration with the existing FAB stack.

Existing framework was also lacking any comparison mechanism and did not provided any rating systems to rate the API aspects. This research provided a comparative analysis mechanism and applied it on two major SD-WAN vendors.

Second Question

"In what ways are leading SD-WAN vendor's APIs alike?"

Second Question Answer

The REST API has become the standard for SD-WAN leading vendors. Due to its serverless nature and requiring less resource REST API are adopted by all the vendors. Other similarities include using JSON as message format.

Third Question

"How could the competence of an SD-WAN vendor's API be evaluated in comparison with another vendor?

Third Question Answer

The competence of an SD-WAN vendor's API can be evaluated by applying the framework suggested in this research which provide a method to do comparative analysis of two or more vendors to measure the competence of the APIs.

This testing will use two variables one to gauge the value scored by specific vendor in that specific aspect. The value of Variable S was given value 1-10 based on testing results. 2nd variable W used in this testing for Weightage of each aspect from the tester perspective. The weightage variable is to give the flexibility to the tester in framework to change the weightage of any variable that is not critical for their integration service provider elements as what each service provider use, and integration entities may be different.

By assigning different score and weightage values a framework was designed and used in this research to evaluate competence of an SD-WAN vendor's API compare to other vendors.

5.3 Discussion on Findings

Based on the results and the findings presented in the above chapter where Cisco Meraki API scored 35 out of 43 and outscored VMware SD-WAN API's. Both vendors claim API as one of the major features of their product and on paper both are REST APIs but there is shear difference in how these APIs are implemented by both vendors. Cisco focus on API development was evident where it scored higher than VMware in each aspect we included in the framework.

This research added two important aspects to API testing framework which were community and documentation.

Community serves as thriving space for the APIs which when implemented and maintained well can serve as win-win space for both SD-WAN vendors and the API users. From API users they can ask queries, get inspired by other developer ideas and request for help when required.

Documentation serves as starting point for any new developer to get use to with the APIs and provide what and what not to do for using the API interface.

Following section go through each aspect of the framework applied and discussed the results and findings of each aspect.

Security

As SD-WAN controller usually reside on the internet, the security of the API interface was treated as most important aspect in this research. It was given highest Weightage of 1 for the same reason.

It was evident when testing the security aspects that two selected vendors have different approach to the security. The first vendor Cisco Meraki required a Key value to authorise and each in request, manually generated and assigned key needed to be present in the header to authorise the request. The Cisco further reduce the attack vector by providing users option to configure source IP addresses to restrict the access of API interface to specified IP address ranges. Cisco also provided the protection against the legitimate users who do have authorised key by rate-limiting the requests. One of the improvements for Cisco on security front recommended by this research is to log each and every API call for audit and forensic purpose.

Compare to Cisco the VMware approach to security was comparatively lack luster and had many areas for improvements. First the cookie-based authentication and authorisation is flawed and can be exploited. Secondly as server need to maintain cookies for each client which make this approach resource intensive on the server. Further to that there was no source address limitation available which make this stateful nature more exploitable and expose the API to whole internet without providing a way to control the attack vector. VMware also not provide any documented way to protect the controller from legitimate users who have been authorised as there was no rate-limiting documentation available about the API. The collected data also showed that there were no logs available to the users on usage of the API which make the audit and any forensic more difficult. All this flaws in the VMware VeloCloud APIs shows that VMware has lot to catch up and improve on the security front.

Performance

Performance includes responsiveness, availability, and scalability of the API interface. The responsiveness and scalability testing required stress testing of the API. As this research did not performed stress testing of the API interface low weightage of 0.1 was given to performance. For performance it was tested if any performance improvement mechanism has been provided by the vendors like paging and cacheing.

Cisco Meraki API do provide paging where subset of the information is requested in the first call and rest of the information in subsequent calls. Cisco Meraki can further improve the performance by providing cacheing mechanism for frequently used API calls.

For VMware case both paging and caching were missing and VMware can improve their API performance by adding both mechanisms.

Integration

Integration covers the vendor maintenance of their changelogs and API versioning. Along with versioning backward compatibility of the version when there is change in the version is also important for Telcos integration as Telcos are usually slow to response to changes.

Cisco API changelog and versioning were well documented. Cisco Meraki also did not depreciate the old version straight away and date of depreciation was publicly available on Meraki website and two years' timeframe was given between announcement of new major version and deprecation date of the older version of the API. This gives ample amount of time for Telcos to adopt the newer API version.

VMware on the other hand did not published any versioning and did not publicly maintained changelog to their API interface. For the case VMware there is room for lots of improvement if we compare it with Cisco API. To be specific VMware need to make their version control and change log of the API interface publicly available.

Usage & Telemetry

Usage and telemetry cover the statistics of overall API usage and information about what APIs calls are frequent and data on usage of the API by specific user and meta data about across their platform by all the users.

Both the vendor needs to improve on usage and telemetry front and usage statistics were missing by both vendors where Meraki Cisco provide usage telemetry for API that is when was the last request made by specific Key and when was key generated. Compare to that VMware only provide stats about key creation. There is high chance that both vendors may be collecting and storing usage statistics internally, but these statistics were not publicly available to users.

Developer Experience

Developer experience was about how easy it is made by vendor for developer to adopt the API and use it for integration. Its very subjective and covers a lot of aspects but one aspect which is measurable is standardisation of the API interface. The more an API follow the standard development guidelines the easier it is for developer to adopt the API and use it for systems integration. As research tested the aspects but did not did actual integration with the telco FAB

stack the low weightage of 0.2 was assigned to developer experience while score was assign based on research experience in adopting the API interface for this testing.

Cisco Meraki API adoption for testing was done without experiencing any issues in adoption as Cisco provide Postman library for adoption and testing. Further Cisco API were following standard guidelines for restful API calls. Response to API calls was as per their documentation.

VeloCloud also provided all the APIs library for postman adoption and testing. Additional time was spent to understand the API calls for VMware compare to Cisco Meraki as VMware VeloCloud was not following the standard guidelines. The major issue with the VMware was that each call was POST call regardless if it was even to get an information from the VMware API interface. Which is against the basic guideline of Restful APIs and resulted in additional time consumed in understanding how these calls are working. Setting API authorization also required multiple attempts as instead of using Key value pair VeloCloud was using additional keyword “Token” in the token value field of the header.

The VMware need to improve its API calls by sticking to standard guidelines for APIs to reduce the time required to adopt VeloCloud API interface and to improve developer experience.

Error Handling

The error handling helps when things not going as per plan. During the integration when requests start failing, there is nothing more painful for developer and integrators then receiving a generic error every time the request fails as developer need vendor help here to identify why the request was failed. There can be thousands if not less different reasons for a request to fail and without proper and meaningful error message it is difficult to pinpoint the reason for failure. Error handling help to pinpoint the reason for the failure and provide a way to preemptively create codes to overcome and resolve that error within the API calls.

The research showed that Cisco Meraki API interface was using standard HTTP status codes for both successful and failed attempts. For most of the instances response status code was good enough for understanding the root cause for an API call to failed. Apart from response status code, the body of the response had a detail description and error message about why request failed.

Research showed that VMware also respond to errors with standard HTML status codes but there is no documentation publicly available on what error code is sent when request failed for specific reason. This made users relies on vendor support to get help when request failed, adding additional steps and time and resources consumed for such support calls to vendor. The research observed that Vmware has lot to improve on error handling and their documentation about error code should be publicly available to API users and developers.

Community

Community aspect has been added by this research to API testing framework. An active API Community shows latest adoption of API by the users and secondly provide a platform for API users to communicate and exchange ideas. Community portal is also good to get customer feedback on what is missing in the API interface and how vendor can further improve the API.

Research showed that Cisco was heavily investing in keeping the API community active and there was reward and incentive mechanism implemented on Cisco community page to reward active community users. There was separate section for API on Cisco Meraki community pages and there was lot of focus on responding to queries by community members as Cisco Meraki API developers were seen responding to queries on API pages. The Community was also used by Cisco to give sample code snippet for adoption and integration of APIs.

Research showed that VMware has lot to catchup on the community front as VMware VeloCloud has not invested mush time and effort on community portal. Although community portal did exist, there was no encouragement by vendor to attract users to its community portal. To start with there was no separate community portal for API usage. There was a community portal for overall usage of VMware VeloCloud SD-WAN solution. Even if we compare the whole community of VMware SD-WAN solution the activity on the VMware complete SD-WAN solution community was lower than the activity on just Cisco Meraki API usage community section.

Documentation

Documentation is 2nd aspect has been added by this research to API testing framework. This research gauge documentation on its comprehensiveness and as well as accuracy. API documentation touch all other aspects of API testing framework as it serves as instructions on how to use the APIs. The good API documentation on one side it helps user to easily consume API interface and on the other hand it saves vendor time and cost on not to answer the integrator queries where a good documentation could serve the purpose. This research has given high weightage of 0.9 to documentation aspect of the framework due to its importance which is 2nd highest weightage in this research.

The research showed that Cisco provided a complete documentation in each aspect including versioning, error handling, detail examples and changelogs. Furthermore, what missing was covered via the community aspect where Cisco was heavily investing in active and large community.

Compare to Cisco Meraki API documentation, research showed that VMware VeloCloud was lacking in its public documentation and need to improve in all aspects of documentation including error handling, change log, versioning, examples, and sample codes for using APIs.

Discussion on Overall results

Below table list down the overall results of the framework application on two vendors Cisco Meraki and VMware VeloCloud

Aspects	Weightage (0 – 1)	Cisco Meraki score (1 – 10)	Cisco Meraki weighted score	VMware VeloCloud score (1 – 10)	VMware VeloCloud weighted score
A1 – Security	1	8	8.0	4	4.0
A2 – Performance	0.1	7	0.7	4	0.4
A3 – Integration	0.4	9	3.6	2	0.8
A4 – Usage & Telemetry	0.3	3	0.9	2	0.6

A5 – Developer Experience	0.2	9	1.8	3	0.2
A6 – Error Handling	0.9	9	8.1	4	3.6
A7 – Community	0.5	8	4.0	2	1.0
A8 – Documentation	0.9	9	8.1	3	2.7
Total			35.3		13.3

Table 6: Vendors weighted scores

Below is the weighted score of both the vendors based on applied testing framework

Cisco Meraki API scored a very high weighted score of 35.3 from possible total of 43

VMware VeloCloud API scored a lackluster weighted score of 13.3 out of 43

On SD-WAN front both are leading SD-WAN vendor infect as per Gartner report VMware VeloCloud is listed as the highest rated vendor for SD-WAN [15]. When we take the API aspect of the vendors Cisco come out as clear winner and VMware VeloCloud has lot to catch up. SD-WAN is lot more than just API but if we compare just based on API VMware is way behind and have a lot of rooms for improvements as based on this research and application of framework not only the overall weightage score of Cisco was high but also in each aspect Cisco Meraki achieved comparatively higher weighted score compare to VMware VeloCloud. The only aspect where Cisco Meraki need significant improvement is usage and telemetry even in that category Cisco Meraki recorded higher weighted score than the VMware VeloCloud.

The result showed that the research provides an API testing framework, which was able to differentiate between wo vendors and was able to assign weighted score which differentiate between a good implementation of API and a bad implementation on the API front.

With the help of Weightage (W) and score (S) variables this framework is flexible to adopt to any tester or solution architect requirements and can be used to do comparative analysis of any two or even more SD-WAN vendors.

The Weightage (W) flexibility is important as different testers and solution architects may have different requirements and some of the factors are not that important for that solution architect. For example, a solution architect testing an open API where API is to consume publicly without any authentication for such, publicly open APIs security may not be that high concern and solution architect can reduce security weightage by lower weightage to security aspect. Another example is about the solution architect who want to adopt the API for very low use case and performance of the API is not critical for the solution architect or may not even be relevant from solution architect perspective. In that case solution architect can assign a weightage of 0 to that aspect to remove influence of unwanted aspect to his testing.

5.4 Conclusion

The proposed framework was able to differentiate between the SD-WAN vendor's API and was able to provide a comparative analysis of each and every aspect of the framework and final score on how a vendor API is comparatively more useful for integration with service provider.

6.0 Conclusion

This chapter provide the overall review of the thesis, what contribution this thesis provides in the field of SD-WAN and API comparisons, what are the limitation of the research work and what future research is required for further advancement in this research. The chapter end with the conclusion of the research.

6.1 Conclusion Summary

The primary purpose of the research was to explore if we can come up with a framework to do comparative analysis of SD-WAN vendor APIs and can we apply this framework to existing API functionality of SD-WAN vendors. In the study we developed the framework consisting of eight different aspects of APIs that needed to be tested to compare the APIs capabilities. We defined a structure to score these aspects under a flexible framework where flexibility was provided by providing capability to researcher to assign different weightage to each aspect based on the researcher requirements. In the research we assigned those weightages and were able to compare two different SD-WAN vendor's API capabilities. The research suggested that the framework can be used for to compare any SD-WAN vendor API capabilities and to find which vendor will be better fitted for the service provider integration as evident in the overall results in section 4.3 where framework was applied on two leading vendors.

6.2 Limitations

This section covers some of the limitation of the research which might have possibly affected the conclusion of the research.

Researcher Bias

As all aspects were decided by the researcher along with the weightage and score assignment, there can be possibly a factor of researcher bias. Although the explanation of each assigned scored are given in the research and reason for each aspect added to the framework but still there can be possibility a researcher bias and counter argument might be present to change the score of the vendor. Regardless the framework is still valid.

Limited number of Vendors

In the research only two vendors are chosen to test against the framework. To further validate the framework, it needed to be applied against additional vendors provided API. The assigned score in the framework can further be refined when framework is applied on the results obtained from multiple vendors.

Performance Testing of the API.

This research did not conduct any performance testing of the API. Performance testing is done using stress testing of the API using multiple calls some time tens of thousands of calls running simultaneously. The performance aspect of the API covers the scalability and availability of the API calls. Scalability is how many simultaneous calls an API interface can handle while availability is basically uptime of the API interface to serve the API calls. Although research covered the performance improvement mechanisms adoption by the vendor. Such performance improvement mechanism includes the paging and cacheing mechanisms. Due to lack of performance testing, the research lacks any concluded results how the API interface for a vendor will perform when controller is responding to high number of API calls.

Integration with Service Provider Stack

One of the limitations of the research is that although it tested all the aspects of the API, the research did not include any actual integration with existing FAB stack as that will involve working with a service provider itself. An actual integration with FAB stack will prove the finding of the research but that will need access to service provider stack which is not publicly available.

Date and time of the research

One of the limitations of the research is that the results produced by the research are applicable on the currently available APIs on the vendor. Vendor may release a newer version of the APIs our coming any shortcoming into their tested APIs in that case the framework will required to be re-applied by taking changes and latest API version into the account.

6.3 Future Work

The research managed to answer the research questions but there are many additional questions are raised by this research which required further study and research.

Additional Vendors

The future research can include using this framework on additional large number of vendors to get similar comparative results from those vendors. As this research only applied the testing framework to two vendors, applying the same framework to large of vendors will further prove the usefulness of the framework and will be able to test the framework application with more effectiveness as additional vendor may results in different scores for each vendor and scoring variable (S) value assignment can be more effective where results from multiple vendors are available to give more validity to score.

Applying framework in other fields API testing

One aspect of future research would be to apply same framework on other fields APIs that is those APIs which serve different purpose example switch APIs or wireless controller APIs to see if the framework is generic enough to use against any APIs. Although the research only developed this framework for SD-WAN vendor API testing, there is no apparent reason this framework cannot be applied on other APIs as the framework doesn't have any specific SD-WAN aspect to it and any would be applicable to any vendor or platform APIs, but this needed another research work to prove its application on any other field.

Performance Testing

One of the future research can be doing actual performance testing using API testers which can generate tens of thousands of calls to test the performance of the controller against high number of calls. The large number of API calls will also bel to test and confirm scalability of the vendor solutions. It will also prove the effectiveness of any rate-limiting done by the vendor to validate that rate-limiting come into account when high number of API calls are made. The resulting rate limiting will also provide the testing mechanism of vendor error handling as appropriate error message should be returned by the controller if error handling is done correctly by vendor to identify the calls failed due to application of rate-limiting.

Actual Integration of Service Provide FAB stack

The research provides framework of usefulness of API for integration with service provider FAB stack and applied this framework on two leading vendors but did not actually integrated the vendors with FAB stack which can be area of further study to further prove and validate the vendors recorded weighted score to see if the actual integration validates the results or not.

Researcher Biasness

One of the limitations of the research is researcher biasness, where further studies can be done to involve a panel of expert to get their inputs on the selected aspects and to have their views and inputs on the assigned scores for each aspect. This will help to remove any researcher biasness possibility from the research. A qualitative result with panel of other experts in the same field will give validity to research on multiple front one if the selected aspect provides complete API testing framework, does additional aspects of community and documentation added by this research were necessary and finally if the expert panel agreed with the assigned score or not.

7.0 References

[1] P. Mohapatra, “Software Defined WAN Overview,” 2018.

[2] P. Wadhwani and S. Kasnale, “Global SD-WAN Market Insight,” *Global Market Insight*, May 15, 2020. https://www.gminsights.com/pressrelease/software-defined-wide-area-network-sd-wan-market (accessed Dec. 23, 2020).

[3] S. Paul and R. Kumar, “SDN: the network of the future,” *CSI Transactions on ICT*, vol. 8, no. 1, pp. 29–32, Mar. 2020, doi: 10.1007/s40012-020-00278-4.

[4] B. Butler, “SD-WAN: What it is and why you’ll use it one day: EDS Live,” *Network World*, Nov. 02, 2016. https://eds-a-ebscohost-com.ezproxy.aut.ac.nz/eds/detail/detail?vid=0&sid=164b3420-5ff6-40e6-a416-7d5ea41c866e%40sessionmgr4008&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=112909390&db=bth (accessed Dec. 24, 2020).

[5] M. Henning, “API design matters,” *Communications of the ACM*, vol. 52, no. 5, pp. 46–56, May 2009, doi: 10.1145/1506409.1506424.

[6] S. Preibisch, *API Development*. Apress, 2018.

[7] S. Vargas, “Harnessing Hybird Deployment,” *Computer Weekly*, 2018.

[8] T. Vijayakumar, *Practical API Architecture and Development with Azure and AWS*. Apress, 2018.

[9] V. Jain, V. Yatri, Kanchan, and C. Kapoor, “Software defined networking: State-of-the-art,” *Journal of High Speed Networks*, vol. 25, no. 1. IOS Press, pp. 1–40, 2019, doi: 10.3233/JHS-190601.

[10] M. Sneps-Sneppe and D. Namiot, “Metadata in SDN API for WSN,” Sep. 2015, doi: 10.1109/NTMS.2015.7266504.

[11] C. Janz, L. Ong, K. Sethuraman, and V. Shukla, “Emerging transport SDN architecture and use cases,” *IEEE Communications Magazine*, vol. 54, no. 10, pp. 116–121, Oct. 2016, doi: 10.1109/MCOM.2016.7588279.

[12] J. Wang, X. Bai, H. Ma, L. Li, and Z. Ji, “Cloud API Testing,” in *Proceedings - 10th IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2017*, Apr. 2017, pp. 385–386, doi: 10.1109/ICSTW.2017.71.

[13] J. Wang, X. Bai, L. Li, Z. Ji, and H. Ma, “A Model-Based Framework for Cloud API Testing,” in *Proceedings - International Computer Software and Applications Conference*, Sep. 2017, vol. 2, pp. 60–65, doi: 10.1109/COMPSAC.2017.24.

[14] Isha, A. Sharma, and M. Revathi, “Automated API Testing,” in *Proceedings of the 3rd International Conference on Inventive Computation Technologies, ICICT 2018*, Nov. 2018, pp. 788–791, doi: 10.1109/ICICT43934.2018.9034254.

[15] B. Analysts Jonathan Forest, A. Lerner, and N. Singh, “Magic Quadrant for WAN Edge Infrastructure,” 2020.

[16] C. Preimesberger, “Top SD-WAN Providers in 2020: EDS Live,” *eWeek*, Aug. 20, 2020. https://eds-a-ebscohost-

com.ezproxy.aut.ac.nz/eds/detail/detail?vid=3&sid=7a7813e8-69f6-4666-b20e-f702b9a37b1b%40sessionmgr4006&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=145263795&db=anh (accessed Dec. 24, 2020).

[17] A. Rodriguez, “RESTful Web services: The basics Develop skills on this topic,” 2008.

[18] S. K. Sood, “Cookie-based virtual password authentication protocol,” *Information Security Journal*, vol. 20, no. 2, pp. 100–111, 2011, doi: 10.1080/19393555.2011.560924.

[19] N. Jovanovic, E. Kirda, and C. Kruegel, “Preventing cross site request forgery attacks,” 2006, doi: 10.1109/SECCOMW.2006.359531.