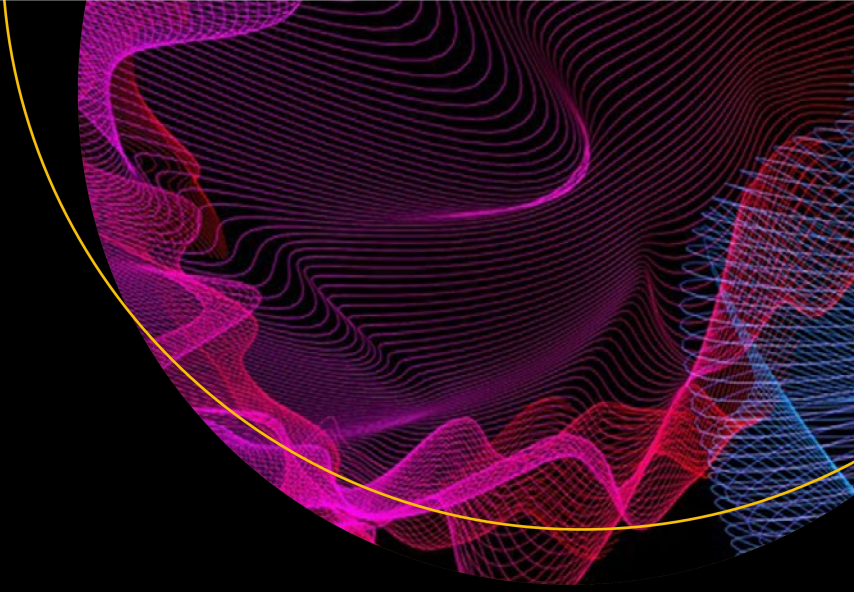

Learning SD-WAN with Cisco

Transform Your Existing WAN Into a Cost-effective Network

Stuart Fordham

Learning SD-WAN with Cisco

Transform Your Existing WAN Into a Cost-effective Network

Stuart Fordham

Apress®

Learning SD-WAN with Cisco

Stuart Fordham
Bedfordshire, UK

ISBN-13 (pbk): 978-1-4842-7346-3 ISBN-13 (electronic): 978-1-4842-7347-0
https://doi.org/10.1007/978-1-4842-7347-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aditee Mirashi
Development Editor: Laura Berendson
Coordinating Editor: Aditee Mirashi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-7346-3. For more detailed information, please visit http://www.apress.com/source-code.

To my wife.

Table of Contents

About the Author

Stuart Fordham, CCIE 49337 is the Network Manager and Infrastructure Team Leader for SmartCommunications SC Ltd, the only provider of a cloud-based, next-generation customer communications platform. Stuart has written a series of books on BGP, MPLS, VPNs, and NAT, as well as a CCNA study guide and the *Cisco ACI Cookbook*. He lives in the UK with his wife and twin sons.

About the Technical Reviewer

David Samuel Peñaloza Seijas works as a Principal Engineer at Verizon Enterprise Solutions in the Czech Republic, focusing on Cisco SD-WAN, ACI, OpenStack, and NFV. Previously, he worked as a Data Center Network Support Specialist in the IBM Client Innovation Center in the Czech Republic. As an expert networker, David has a wide diapason of interests, while his favorite topics include data centers, enterprise networks, and network design, including software-defined networking (SDN).

Acknowledgments

I'd like to say thanks to the following:

My wife for her encouraging words: "*Well, if you played the guitar less, you'd have that book finished by now*." It's finished now. So back to playing the guitar!

My boys for being patient while "Daddy is working *again*." You guys are amazing.

David Peñaloza: The best technical editor a guy could ask for.

The team at Apress, for giving me another shot after I had finished freaking out about my workload.

My team at SmartCommunications. You guys are the best.

Introduction

Learning SD-WAN with Cisco explores what SD-WAN is and how it will benefit modern networks and builds an example network.

This book covers the evolution of modern networks and how the software-defined wide area network (SD-WAN) has risen to the forefront. We will explore the components of SD-WAN for orchestration and management and look at the edge devices and then go on to build a network from the ground up, deploying cEdge and vEdge routers. We will apply policies and templates to manage the control and data planes as well as VPNs, Internet access, security, and quality of service.

We will also explore reporting and management, along with upgrading and troubleshooting.

This book is intended for those who would like to get an understanding of what SD-WAN is and how we deploy, configure, manage, and troubleshoot it.

CHAPTER 1

An Introduction to SD-WAN

In this chapter, we are going to look at what SD-WAN is and how it came about.

The Traditional Network

The networks we rely on for both business and pleasure on a day-to-day basis are susceptible to many factors that can result in a slow and unreliable experience.

We can experience latency, which either refers to the time between a data packet being sent and received or the round-trip time, which is the time it takes for the packet to be sent and for it to get a reply, such as when we use ping.

We can also experience jitter, which is the variance in the time delay between data packets in the network, basically a "disruption" in the sending and receiving of packets.

We have fixed bandwidth networks that can experience congestion: with 5 people sharing the same Internet link, each could experience a stable and swift network, add another 20 or 30 people onto the same link and the experience will be markedly different.

There are ways we can help manage the experience for all. We can implement quality of service (QoS), which we can use to prioritize traffic, such as voice and video, where fluctuations in the network due to these factors are noticeable. We can also use QoS to give each user their fair share of bandwidth and to ensure that the right amount of bandwidth is assured for our mission-critical applications.

QoS works well within the boundaries of the network but requires manual intervention. We need to know what our traffic is, where that traffic needs to go, and what our priority traffic is. We can help it get there faster and with a larger degree of assured delivery, but it still requires the network administrators to ensure that the paths the traffic is to take are present, working, and reliable. QoS, combined with policy-based routing (PBR), can also provide a way to route traffic out of different interfaces, making use of dedicated high-speed links for mission-critical traffic and other links for user traffic such as Internet browsing and media streaming. These do, again, require the network administrator to plan this traffic splitting out, and this method is not exactly "dynamic."

There are mechanisms to dynamically route traffic though, such as Multiprotocol Label Switching Traffic Engineering (MPLS-TE). Through the use of MPLS, a link-state protocol such as OSPF or IS-IS, RSVP (Resource Reservation Protocol), and CBR (Constraint-Based Routing), we can have a network that learns about changes in the network and reacts by performing path selection in the network.

MPLS-TE is very much in the realm of large enterprises and ISPs, though, and with it comes increased costs in both infrastructure and engineering.

Today, we are, however, deep into cloud adoption, where pretty much everything can now be offered "As a Service." Platform, infrastructure, and software are all deployable at a click of a button, and whole virtual data centers can be stood up in minutes. Names such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud are so embedded in 21st-century technology that it is fast approaching the time where we will soon not even begin to comprehend how we managed before "the cloud."

So how do we marry up the needs of today's cloud computing, the benefits of QoS, and MPLS-TE as well as the dynamism we need for modern networks, while, at the same time, increasing security, reducing costs, and having a technology that is easy to use? These seem like a lot of contradictory criteria to fulfil.

The answer is SD-WAN, or software-defined networking in a wide area network.

SD-WAN

SD-WAN has taken the concept of software-defined networking (within a local area network) and cloud orchestration and applied it to the wide area network.

There are, according to Gartner,[1] four requirements for an SD-WAN:

- It must have the ability to support multiple connection types.
- It should be able to perform dynamic path selection.
- It should have the ability to support VPNs and third-party services (such as firewalls).
- It must have a simple interface.

It is not just Gartner that has put these requirements on paper. This is also the standard defined by MEF (which once stood for the Metro Ethernet Forum) in MEF 70. MEF is an international industry consortium that looks to promote the adoption of assured and orchestrated connectivity services across automated networks. Members of MEF include Cisco, Ericsson, Huawei, Juniper, Nokia Networks, VMWare, and more companies with "telecommunications" or "telecom" in their names than you can shake a stick at.

[1] www.networkworld.com/article/3031279/sd-wan-what-it-is-and-why-you-ll-use-it-one-day.html

MEF 70[2] is not the easiest document to understand. It uses many TLAs (three-letter acronyms) and contains nuggets like this:

> *MEF Services, such as SD-WAN, are specified using Service Attributes. A Service Attribute captures specific information that is agreed on between the Service Provider and the Subscriber of a MEF Service, and it describes some aspect of the service behavior.*

Got that? Great, neither did I. However, if we take an SD-WAN deployment from a more practical angle and put it into some context, it does start to make sense. I will try to translate as we go through the various components.

We start with the SD-WAN edge device. These devices can either be physical ones or virtual appliances. The SD-WAN edge devices need to support multiple connection types, such as MPLS, Internet such as leased lines, and LTE. The edge device is "*situated between the SD-WAN UNI, on its Subscriber side, and UCS UNIs of one or more Underlay Connectivity Services on its network side*," meaning that these devices live at the demarcation point between the business network (Customer Premises) and the ISP, the SD-WAN UNI (User Network Interface), and the Underlay Connectivity Service (UCS), or the Internet circuit.

Because we have an underlay service, it makes sense that we also have an overlay. The overlay is the network we are orchestrating, and we do this through the SD-WAN Controller and the SD-WAN Orchestrator; these devices control our policies concerning application flow and security. The overlay needs to be able to understand the network and feed information back to the edge devices so that they may choose the best paths across the network, as well as controlling our VPNs and other services.

So once we start looking at SD-WAN from a practical standpoint, MEF 70 actually starts to make sense.

---

[2] www.mef.net/resources/technical-specifications/download?id=122&fileid=file1

Cisco, along with Huawei, Nokia Networks, and Verizon, among others, participated in the development of MEF 70, but this was by no means Cisco's first step into the world of SD-WAN.

Cisco and SD-WAN

Cisco had a product called iWAN (intelligent WAN), which provided traffic control and security and integrated into Cisco branch office routers. It offered QoS, WAN optimization, and VPN tunneling, without the cost of expensive MPLS VPNs.

iWAN made a lot of sense, as with the lowering cost of today's Internet links, along with the improvement in their SLAs, MPLS is becoming less attractive. iWAN could provide similar capabilities to MPLS VPN, such as WAN optimization, QoS, and VPN tunneling, all without affecting performance, security, or reliability.

The network overlay used by iWAN is DMVPN (Dynamic Multipoint VPN) and IPSec, which enables the use of any carrier service (MPLS, broadband, and 3G/4G/LTE).

Traffic is routed based on metrics such as SLA, endpoint type, and network conditions. This is achieved using PfRv3 (Performance Routing Version 3), which uses differentiated services code points (DSCP), and an application-based policy framework to optimize bandwidth and path control, protecting applications and increasing bandwidth utilization. PfRv3 looks at the application type, network performance in terms of jitter, packet loss, and delay and can make decisions to forward traffic over the best-performing path.

We can, with iWAN, make networks use MPLS networks for some traffic (e.g., business-critical and VoIP) and other traffic (less critical) use the public Internet, as shown in Figure 1-1.

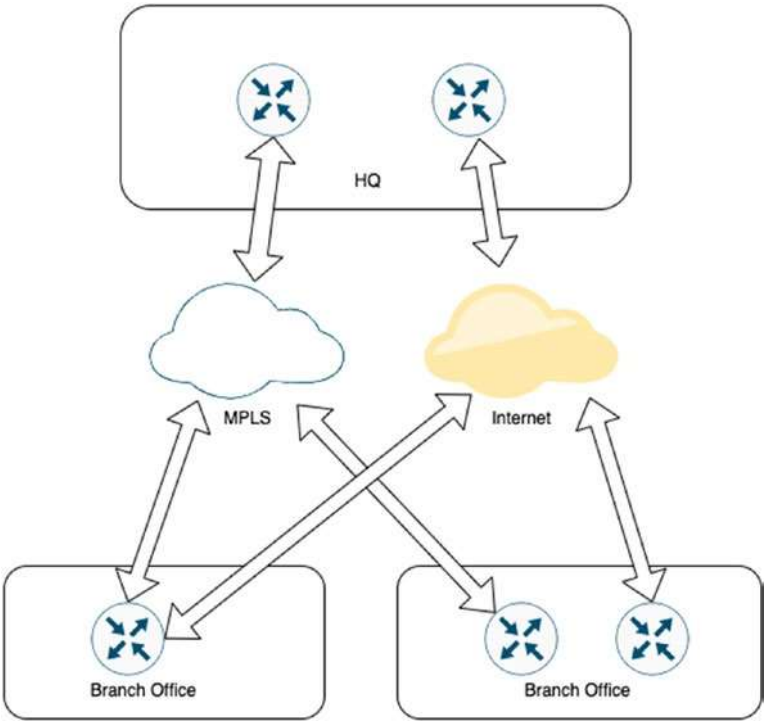

Figure 1-1. *The iWAN network*

With PfR, border routers collect traffic and path information, sending it to a master controller (a dedicated router). The master controller is responsible for enforcing the service policies to match the requirements of the application.

Applications are optimized over the WAN using Cisco's Application Visibility and Control (AVC) and Wide Area Application Services (WAAS). AVC (which includes technologies such as Network-Based Application Recognition 2 [NBAR 2], NetFlow, and QoS) is essential here as many applications use the same ports (such as 443). Spotify, for example, uses a destination port of 4070 for its player, but will use port 443 or even 80 if the former port is unavailable, making the implementation of traffic control

on Spotify impossible when based on the destination port. Because of this reuse of ports, we can no longer rely on static port classification, so AVC uses deep packet inspection to identify applications and to monitor their performance. iWAN also leverages Akamai for branch router caching.

iWAN is secured through IPSec encryption, zone-based firewalling, and ACLs (access control lists), protecting the WAN over the public Internet. It also uses Cisco's Cloud Web Security to provide a proxy to protect users over the Internet and is controlled using the APIC-EM (Application Policy Infrastructure Controller Enterprise Module).

All this sounds great. But why has Cisco rapidly moved to SD-WAN, instead of investing more in its existing product, iWAN?

The simple answer is that while all its benefits made it very attractive, in reality, iWAN was hard to deploy and manage. iWAN is not alone in technologies that have been sidelined, two more of which are PfR and NBAR, which, coincidentally, are two technologies used by iWAN.

The use of APIC-EM, for example, while great for managing iWAN, was only really useful in greenfield deployments. If you already had the building blocks of iWAN in place (DMVPN, QoS, PfR), then rolling out APIC-EM would pretty much require replacing all the existing configurations, which made switching to use the APIC-EM a tough and potentially expensive decision to make.

iWAN is not dead though, far from it. While new customers into the field will be steered toward SD-WAN, the ISR routers that are key to an iWAN deployment hold around 80% of the market share of branch office routers, and each year, Cisco sells around $1.6 billion of ISRs. There is too much invested by Cisco and its customers for iWAN to be ditched completely. However, the focus is now pointed directly at SD-WAN, which has some of the features that iWAN missed, such as an easy-to-use interface, which is, perhaps, why Cisco set its sights on Viptela.

Viptela

Viptela was founded in 2012 by ex-Cisco directors Amir Khan and Khalid Raza. While it was in "stealth mode" and no one (in the general public) knew what it was doing, it received financial backing from Sequoia Capital. Considering the companies that Sequoia has backed in the past, such as Apple, Google, PayPal, YouTube, Instagram, and WhatsApp, Viptela was probably a surefire winner early on.

Over the next couple of years, Viptela emerged from stealth mode into the taking-the-network-world-by-storm mode. It garnered much praise and many customers. It was named several times in CRN's 10 Coolest Networking Startups, named a Gartner Cool Vendor and a Next Billion Dollar Startup by *Forbes*. Not a bad start (for a startup)!

Between 2012 and 2017, Viptela boasted customers such as Verizon, Singtel, and The Gap. Others were used in published case studies but preferred to remain unnamed, which is pretty common in the banking world.

The lure of Viptela is that it offers virtualization of the WAN and is carrier agnostic. Through the WAN overlay technology, communications are secured across whichever medium is used, even broadband and 4G/LTE. Similar to iWAN, in essence, but vastly different in deployment.

The report from clothing retailer The Gap makes for interesting reading and gives the reader a good idea about how and why Viptela was able to make such good ground within such a short amount of time.

The Gap started to roll out SD-WAN in 2015, to alleviate the reliance on the expensive MPLS lines and instead move to the cheaper public Internet. With SD-WAN, they could still do this, as well as keeping the traffic encrypted. Snehal Patel, Gap's network architect, said that they could connect up to 25 or more of their stores per night. Each upgraded store also had between ten and fifteen times the bandwidth it had previously. SD-WAN was also about 50% less expensive than their original method.

It is easy to see the benefits of Viptela's SD-WAN. You get the increased speed, at a lower cost, and rolling it out can be done at a very impressive pace.

You can read the original transcript from the *Wall Street Journal* here: *https://web.archive.org/web/20160726182850/http://blogs.wsj.com/cio/2015/11/05/gap-connects-stores-over-the-internet-with-software-defined-networking/*.

By 2017, Viptela had 16,000+ branch office deployments and proclaimed on its website the following benefits:

- 50% lower costs
- 10x more bandwidth
- 5x cloud performance

With such stores as The Gap putting their success story in the light, it's easy to see why Viptela did so well so quickly.

This brand-new network as a service offered seamless integration with Office 365, Azure, and AWS and used a simple interface (especially when compared to iWAN's APIC-EM). Policies can be used to send latency-sensitive traffic across dedicated MPLS lines and use the "regular" Internet for less critical applications, such as Office 365, and 4G/LTE for remote office where MPLS or broadband is not an option (Figure 1-2).

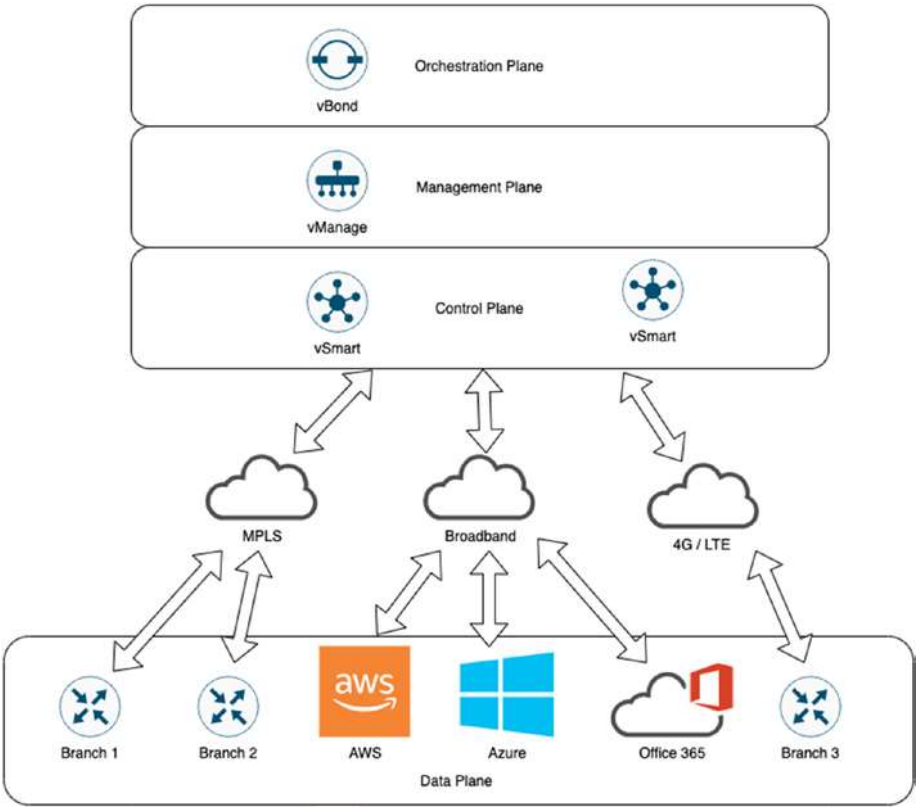

Figure 1-2. *The Viptela SD-WAN*

The difference between Viptela's fabric and Cisco's iWAN is in the ease of connectivity into the likes of AWS, Azure, and Office 365, as well as the simplified management (again, compared to APIC-EM).

It is no wonder, therefore, that in 2017, Cisco bought Viptela for $610 million. Viptela aligned perfectly with Cisco's principles of security, virtualization, automation, and analytics, in their DNA (Digital Network Architecture).

Let's look at the different components that make up the (Cisco) SD-WAN (or SD-WAN Secure Extensible Network [SEN] as they also term it).

Components of a Cisco SD-WAN

From Figure 1-2, you can see that there are four distinct areas to the SD-WAN. At the bottom, we have the connectivity aspect, the data plane. This can be offices or third-party services such as AWS and Office 365. For this connection, we need an "edge device."

vEdge and cEdge

Because of the purchase by Cisco, the Cisco ISR, ASR, and CSR1000v routers are now part of the Viptela ecosystem (subject to running the correct software image); this is in addition to the products already made by Viptela, such as the low-cost vEdge 100b, which can be purchased for under $300.

The edge device takes care of the packet forwarding; they establish the secure virtual overlay network and come in many different flavors:

- vEdge 100b (Ethernet only)
- vEdge 100m (Ethernet and an integrated 2G/3G/4G modem)
- vEdge 100wm (as the 100m but with wireless LAN functionality)
- vEdge 1000 (8 fixed GE SFP ports)
- vEdge 2000 (2 pluggable interface modules)
- vEdge 5000 (4 Network Interface Modules)
- ISR 1100 4G (4 GE WAN ports)
- ISR 1100 6G (4 GE WAN ports and 2 SFP WAN ports)
- ISR 1000 series
- ISR 4000 series

- ASR 1000 series
- ENCS 5000
- vEdge Cloud
- CSR1000v

The vEdge Cloud device can run in VMWare ESXi (5.5. and 6.0), KVM, AWS, and Azure, and it is this image we will be using the most (along with the CSR1000v).

As well as looking after the routing (which can be either OSPF or BGP, or static), the vEdge devices also support AAA, bridging (802.1Q, VLANs, integrated routing-bridging), IPSec, DDOS prevention, NAT (network address translation) traversal, QoS, multicast and policies for routing, application awareness, control and data policies, ACL policies, VPN membership policies, and service advertisement and insertion policies. It also supports the standard set of functions that you would expect from any router, such as SNMP, NTP, DHCP (client, server, and relay), Syslog, SSH, NAT, and PAT.

The vEdge certainly feels like a very well-rounded and fully functioning replacement for most existing router deployments.

The next step-up in the overlay is the control plane, which uses vSmart devices.

vSmart

The vSmart controller controls the data flowing through the network, it is the "routing brain," and once the vBond orchestrator has authenticated the vEdge devices, the vSmart controller manages the connectivity between vEdge devices.

The vSmart devices have a centralized policy engine that looks after the routing data, access control, segmentation, extranets, and service chaining.

There are six main functions provided by the vSmart controller:

- Control plane establishment and maintenance with each vEdge device.

  Each connection is a DTLS tunnel (Datagram Transport Layer Security) carrying the encrypted payload between the vSmart controller and the vEdge router. This information is what allows the vSmart controller to build up a picture of the network, the topology, and for the vSmart controller to perform route calculations to determine the best paths.

- Overlay Management Protocol (OMP). We will look in greater depth at OMP in Chapter 4, but essentially OMP facilitates the overlay network.

- Authentication. Using preinstalled credentials, the vSmart controller can communicate with each new vEdge device as they come online, ensuring that only authenticated devices can connect to the network.

- Key reflection and rekeying. vEdge routers send data-plane keys to the vSmart orchestrator which sends them to other vEdge routers.

- Policy engine. vSmart provides the inbound and outbound policies to look after routing and access control.

- Netconf and CLI. Netconf is used to provision vSmart controllers. Each vSmart controller provides local CLI access.

In Chapter 6, we will deploy a vSmart controller.

vManage

vManage is a Network Management System (NMS) that is used to configure and manage the overlay network. We use this to configure and manage vEdge devices.

Like any good NMS, we can use vManage to create and store the configurations of our network devices. vManage will push down the certificate and configurations onto vEdge and cEdge devices as they come online (this is different in hardware devices, where the certificate is preinstalled on a SUDI or TPM chip during the manufacturing process). It can also generate bootstrap configurations and decommission vEdge devices if required.

We will deploy a vManage server (also referred to as a "controller" or "device" throughout this book) in Chapter 3.

vBond

The vBond device performs functions such as the authentication and authorization of each element in the network, onboarding, and STUN. It is this piece of the topology that tells the rest of the network how they interconnect to the other components, essentially sitting between the vEdge and vSmart devices passing messages for them (initially at least, until the onboarding process has completed).

The functions performed by the vBond are

- Control plane connection. Each vBond orchestrator maintains a DTLS tunnel with each vSmart controller in the network. It also uses DTLS to communicate with the vEdge routers as they come online so that it can authenticate the router and aid the router to join the network.

- NAT traversal. vBond is also responsible for helping vEdge or vSmart devices that sit behind network address translation communicate with the rest of the network, orchestrating the NAT traversal (initially). For this to work, the vBond should live in a public address space.
- Load balancing. Where there are two or more vSmart controllers, the vBond performs load balancing of vEdge routers between the vSmart controllers.

We will deploy a vBond orchestrator in Chapter 5.

Summary

In this, our first chapter, we looked at the driving forces behind the birth of SD-WAN, as well as the different parts that will make up our network. But first, we need to create that network, which we will do using EVE-NG (Emulated Virtual Environment Next Generation).

CHAPTER 2

Deployment Overview

In this chapter, we are going to set up EVE-NG, create our SmartNet account, and download the components we need to make our network. At the end of the chapter, we will look at VMWare and KVM as alternatives to EVE-NG.

EVE-NG

EVE-NG (Emulated Virtual Environment Next Generation) is a virtual platform designed to run many different network devices, as well as Windows and Linux operating systems.

Many years ago there was a bit of software called Cisco IOU (IOS On Unix), and a very kind and clever guy called Andrea Dainese put a web front end on it and called it iou-web. It was very popular. Then he created UNetLab (Unified Networking Lab), which allowed the use of different vendor images.

UNetLab then forked into EVE-NG, which is where we are now.

There are two versions of EVE-NG: the community edition, which is free, and the professional edition. The professional edition, which costs 99EUR, can run up to 1024 nodes and has Docker support. The community version, which is free, can run up to 63 nodes and lacks Docker support, but this is fine for our purposes.

The environment can run on vSphere, VMWare Workstation, or bare metal servers. If you have ever installed a Linux operating system, then it is all very straightforward, and there should be no surprises. For this chapter, I am assuming that you will be installing the server on a hypervisor.

Download the install ISO from www.eve-ng.net/index.php/download/. Again, the community version is more than suitable for our needs.

We need to properly size our server, so give it as much memory and CPU cores as you can. To get a rough idea of what we need, see Table 2-1.

Table 2-1. *Hardware resource requirements*

	vCPU	Memory (GB)	Disk space (GB)
vBond	2	4	10
vManage	16	32	30 + 100GB
vManage(2nd)	16	32	30 + 100GB
vSmart	2	4	16
vEdge	2	2	8
Total	**38**	**74**	**294**

We also need enough ram and CPU to run the actual operating system. While, thanks to hypervisor magic, we can over-provision memory and CPU, you still need something fairly capable.

The network we are going to run was created on a dual Xeon X5675 running at 3.06GHz with 48GB of memory. Running 15 QEMU nodes, CPU usage was around 28% and memory usage ran at around 79%. So there is some wiggle room in what can be run.

Once your VM has been created, boot it up using the ISO image downloaded from the link provided earlier. The first screen will be the Ubuntu installation window (Figure 2-1).

Select the option "*Install Eve VM*" (unless you are doing a bare metal, in which case choose the "*Install Eve Bare*" option).

Figure 2-1. *The initial install screen*

In the next window, choose your language (Figure 2-2).

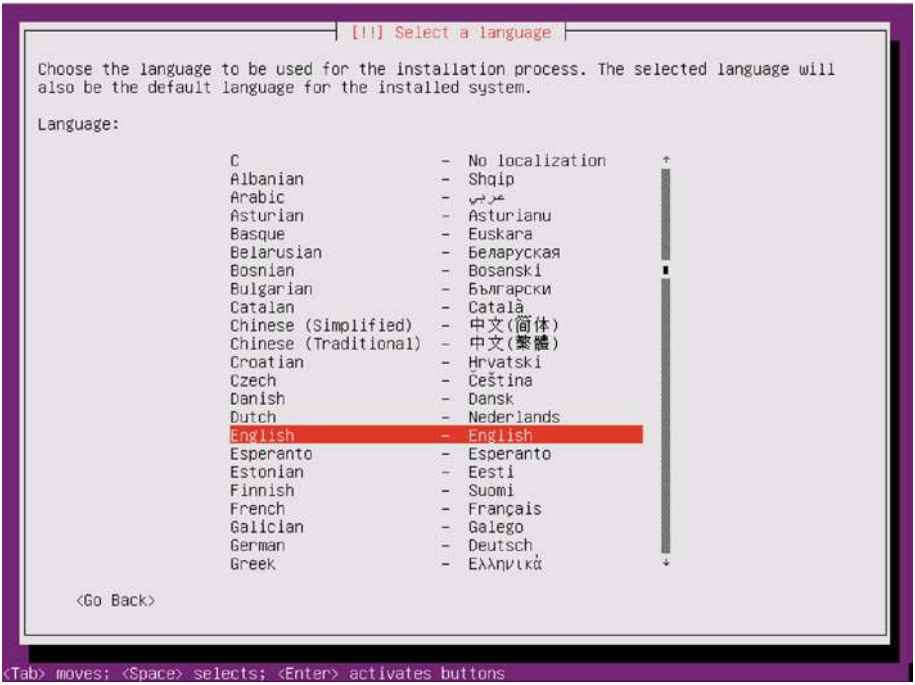

Figure 2-2. *Choosing your language*

Then select your location (Figure 2-3).

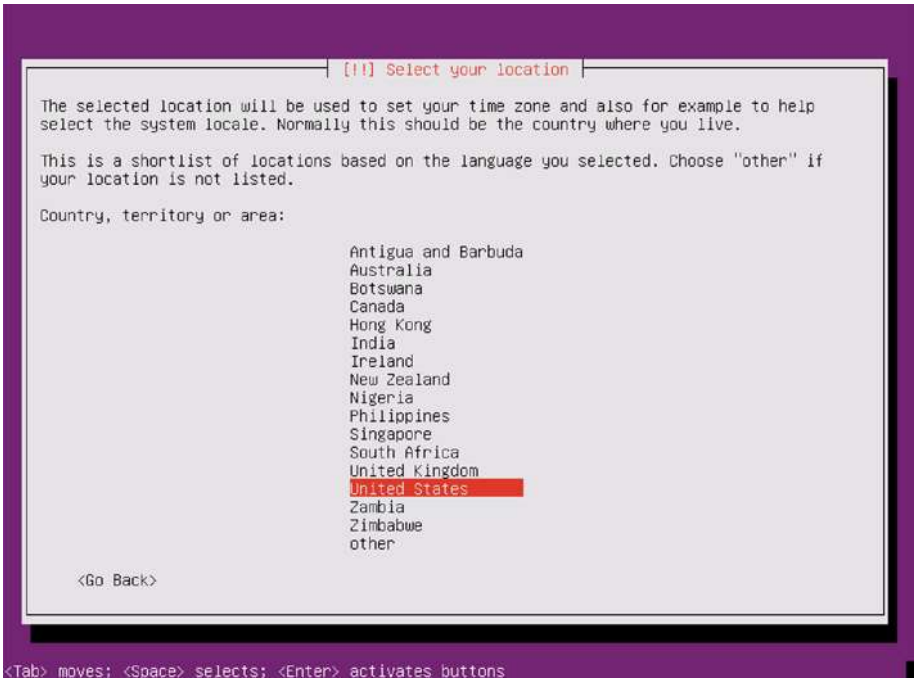

Figure 2-3. *Select your location*

Set the hostname (Figure 2-4).

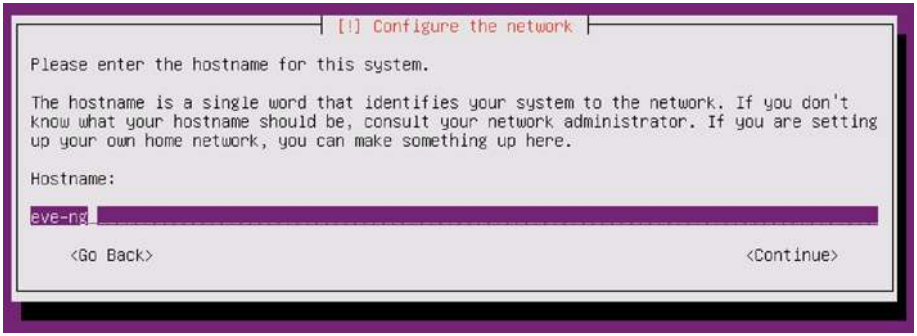

Figure 2-4. *Setting the hostname*

Set the time zone (Figure 2-5).

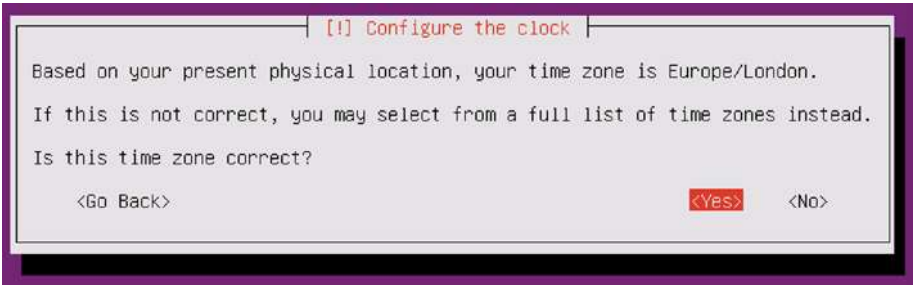

Figure 2-5. *Setting the time zone*

If you use a proxy on your network, then set it in the following window (Figure 2-6).

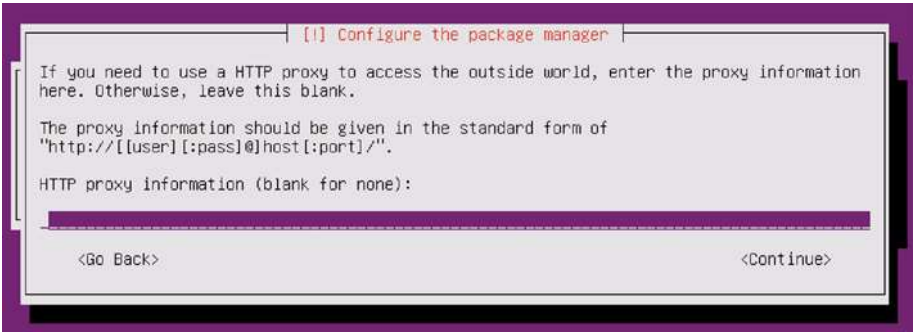

Figure 2-6. *Setting the proxy information*

The installation should complete now, and you should be greeted with the login screen, as shown in Figure 2-7.

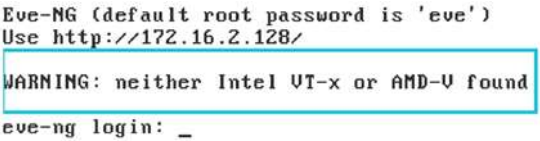

Figure 2-7. *The EVE-NG logon screen*

If you see the same warning as the preceding, then you need to shut down your VM and edit your settings. Don't do this just yet though, as we need to complete the setup.

You will be prompted to enter the root password (Figure 2-8), so enter one of your choosing.

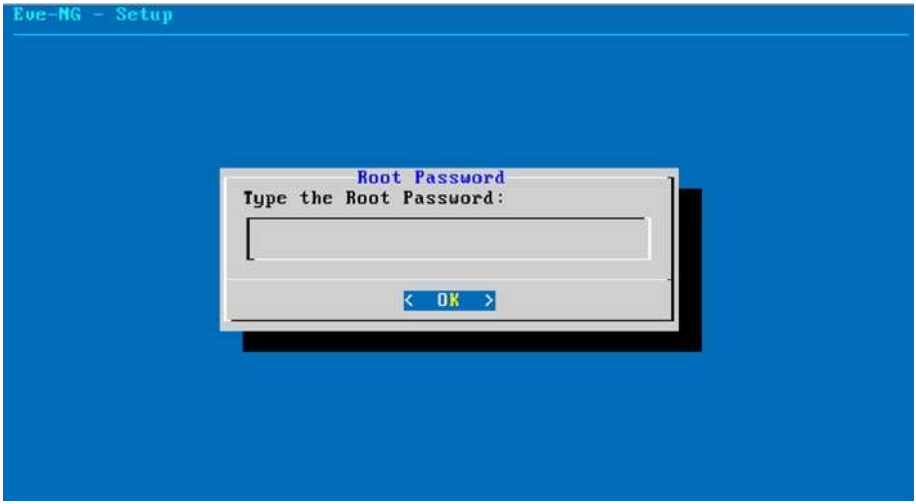

Figure 2-8. *Setting the root password*

Type the password again, when prompted, and then set the hostname (again) (Figure 2-9).

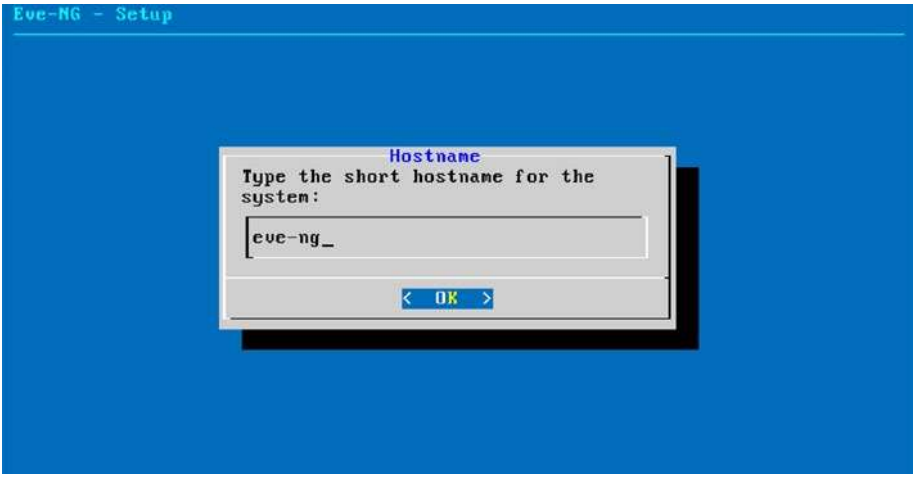

Figure 2-9. *Setting the hostname*

Then set the DNS domain name, if you need to (Figure 2-10).

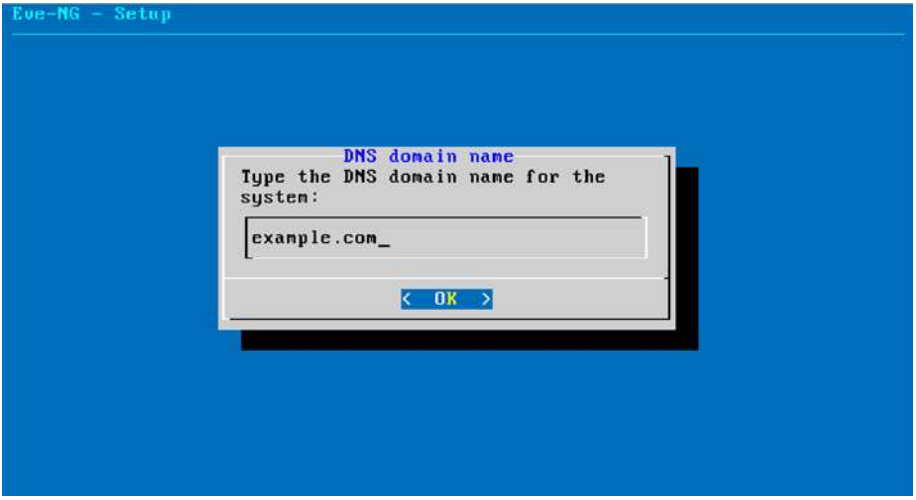

Figure 2-10. *Setting the DNS domain name*

On the next page, we set our networking connectivity, either DHCP or static (Figure 2-11).

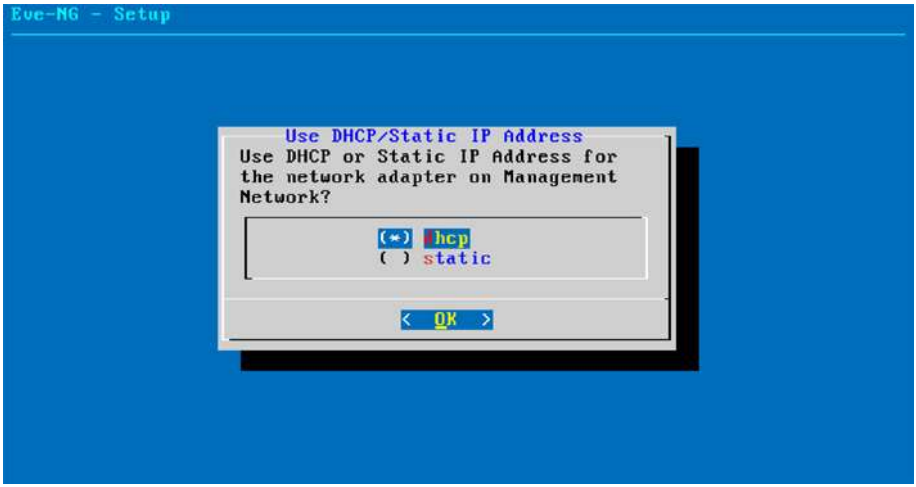

Figure 2-11. *Setting the network adapter*

Set the NTP server, if you have one running in your network (Figure 2-12).

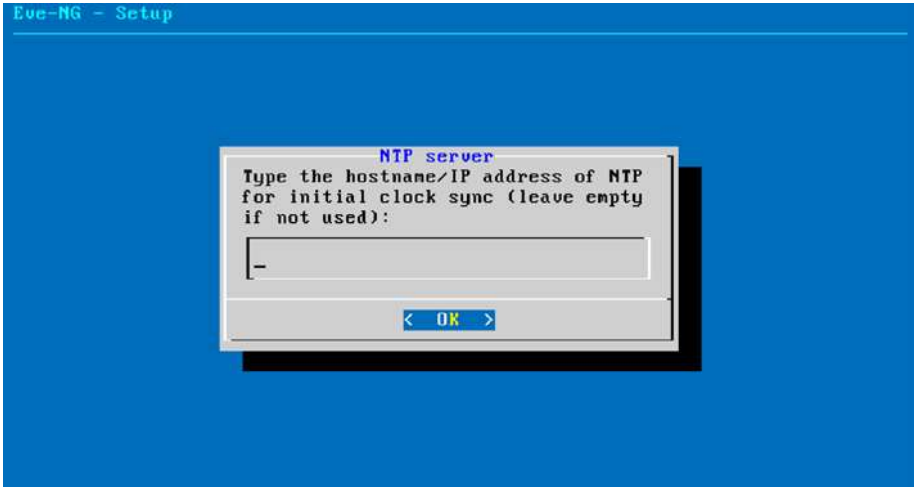

Figure 2-12. *Setting the time servers*

Lastly, set the proxy details (or leave as the default if you do not have a proxy) as shown in Figure 2-13.

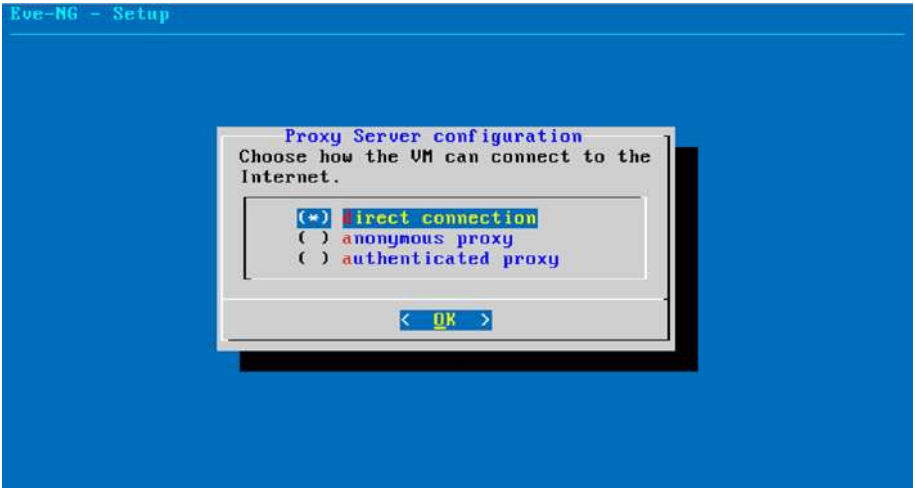

Figure 2-13. *Proxy configuration*

If you did get the warning about "neither Intel VT-x or AMD-V found," then edit your VM settings to enable virtualization support.

In VMWare Fusion, it looks like this (Figure 2-14).

Figure 2-14. *Virtualization support*

Now your login page should look like this after a restart of the virtual machine (Figure 2-15).

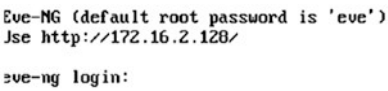

Figure 2-15. *The EVE-NG login screen without warnings*

We are good to proceed.

Getting the SD-WAN Software

We need to download the software from Cisco next. For this, you will need a valid service contract. Your Cisco rep may also be able to help you out here.

Head over to `https://software.cisco.com`, and select "*Manage Smart Account*." Then click Virtual Accounts.

Create a new virtual account. Give it a name, and set it to Private. Click Save (Figure 2-16).

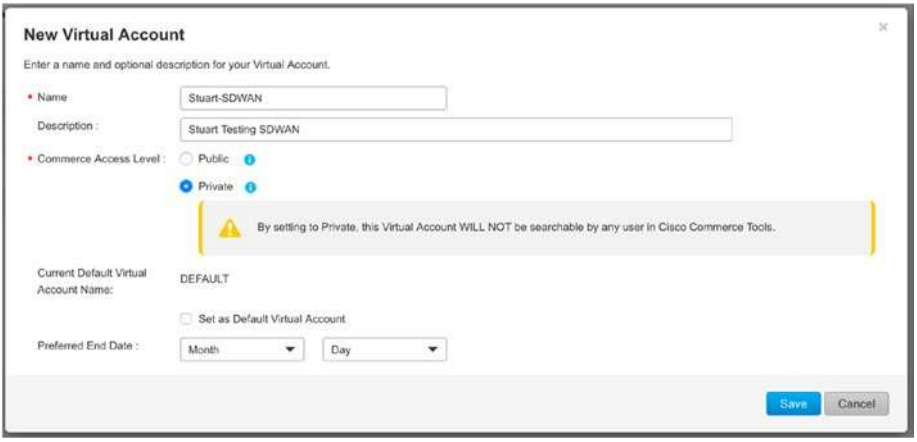

Figure 2-16. *Creating the SmartNet virtual account*

Your virtual account should be visible (Figure 2-17).

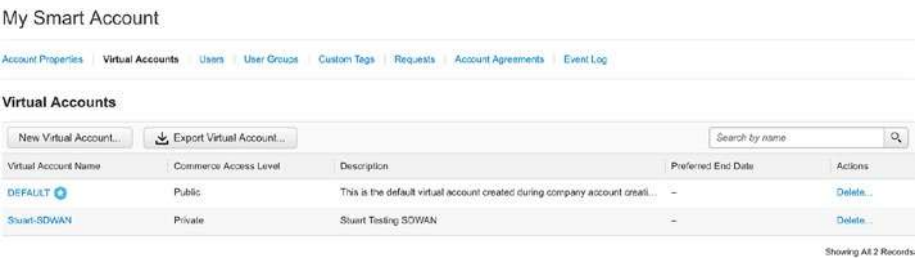

Figure 2-17. *The virtual account has been created*

The next step is to create a vBond controller. Head back to the main software page where we started, and click the "*Plug and Play Connect*" link (Figure 2-18).

Figure 2-18. *The Plug and Play Connect portal*

Select the virtual account you created earlier (Figure 2-19).

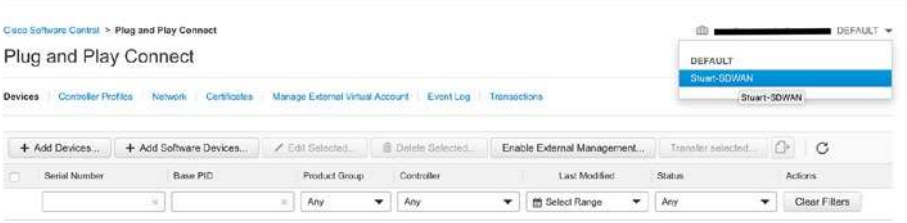

Figure 2-19. *Selecting the virtual account*

Next, click "Controller Profiles" (Figure 2-20).

Figure 2-20. *Controller Profiles*

Click “Add Profile” (Figure 2-21).

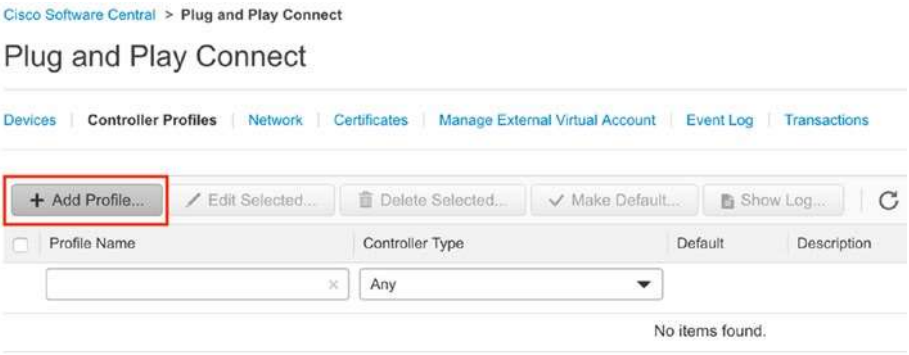

Figure 2-21. *Adding a profile*

Select VBOND from the drop-down menu (Figure 2-22).

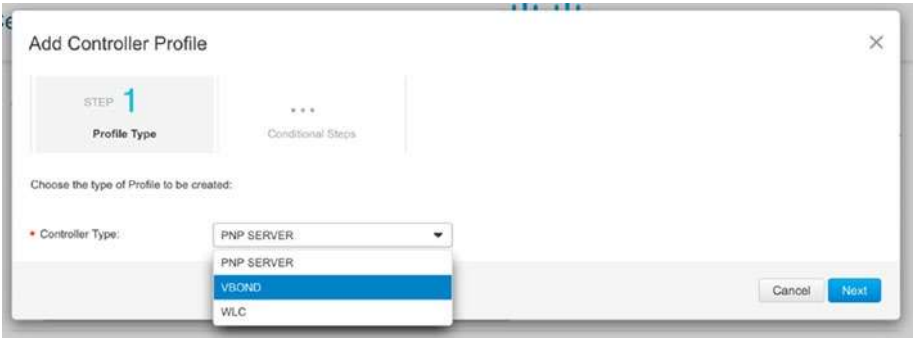

Figure 2-22. *Selecting vBond*

Give the profile a name, and set an organization name. Set the Primary controller to IPv4, and add an IP address. You can use any address you like here (Figure 2-23).

Click Next.

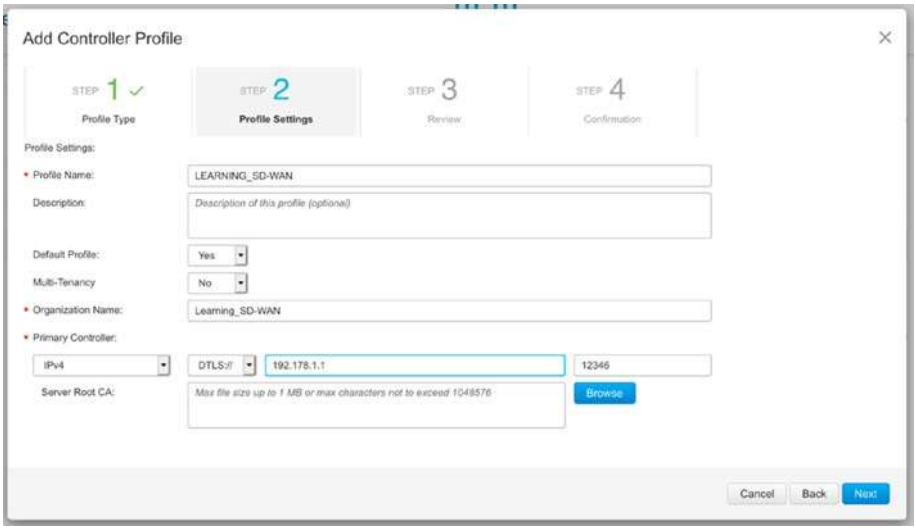

Figure 2-23. *Adding the controller profile*

Confirm your settings, and click Submit (Figure 2-24).

Figure 2-24. *Confirming the settings*

Click Done to confirm the profile (Figure 2-25).

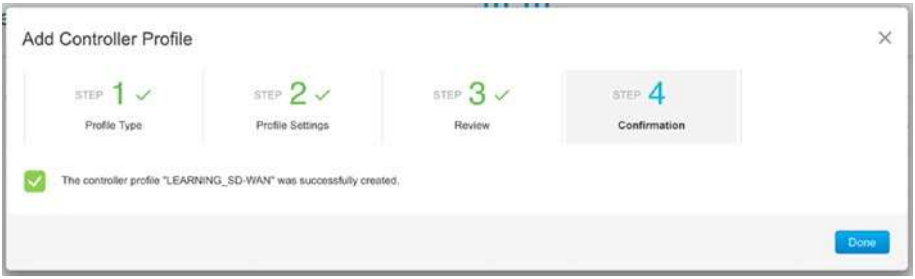

Figure 2-25. *Finishing the controller profile*

You should see the profile that has been created (Figure 2-26).

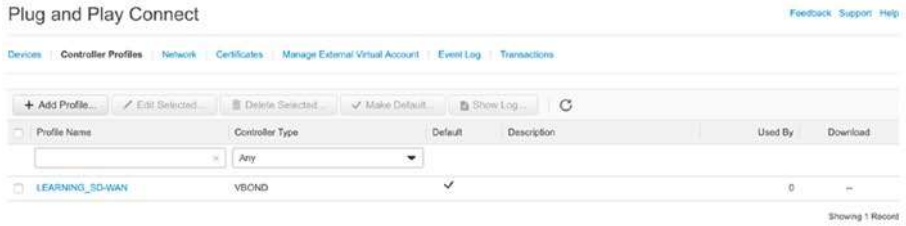

Figure 2-26. *The new controller profile*

Go to Devices (Figure 2-27).

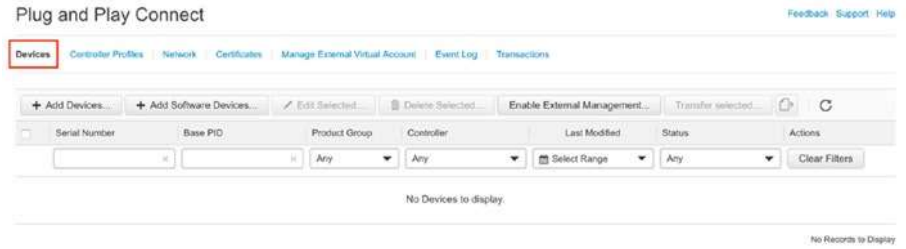

Figure 2-27. *Devices*

Select “Add Software Devices” (Figure 2-28).

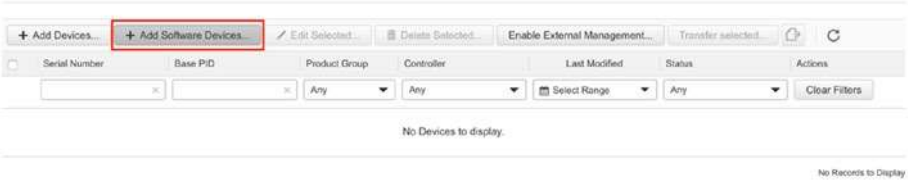

Figure 2-28. *Add software devices*

Click “Add Software Device” (Figure 2-29).

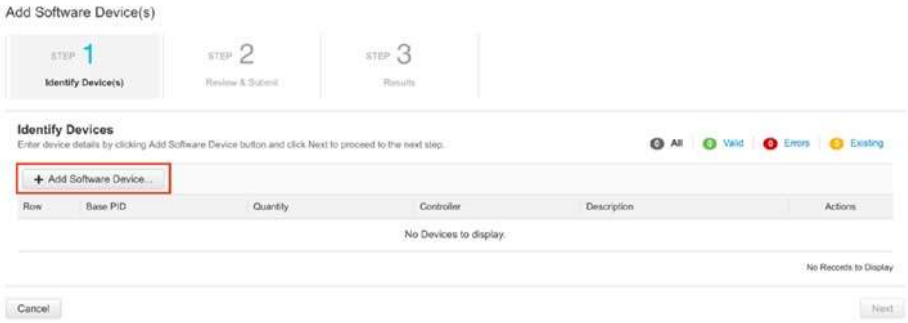

Figure 2-29. *Identifying the devices*

In the Base PID box, start typing “VEDGE-CLOUD-DNA,” and select it from the drop-down when it appears (Figure 2-30).

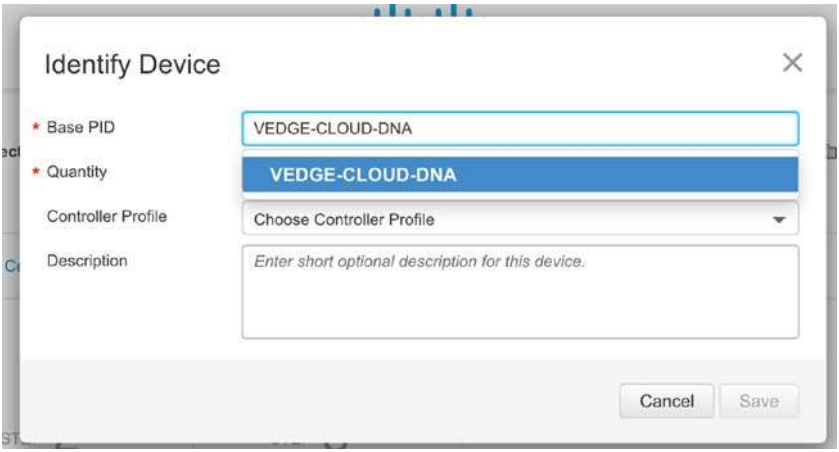

Figure 2-30. *Selecting vEdge*

Set the desired quantity (such as 5 which is a good number; there is a limit of 20 "free" nodes), and from the Controller Profile drop-down, select the profile name we created a moment ago (Figure 2-31).

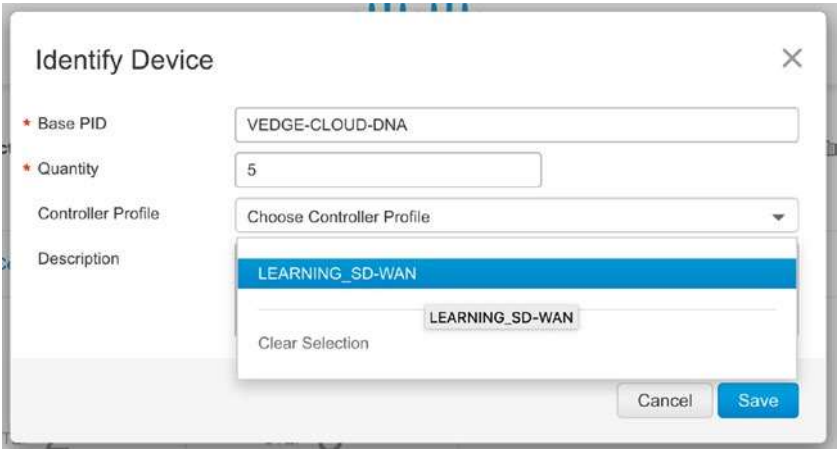

Figure 2-31. *Associating the device to the controller profile*

Click Save, and then click Next (Figure 2-32).

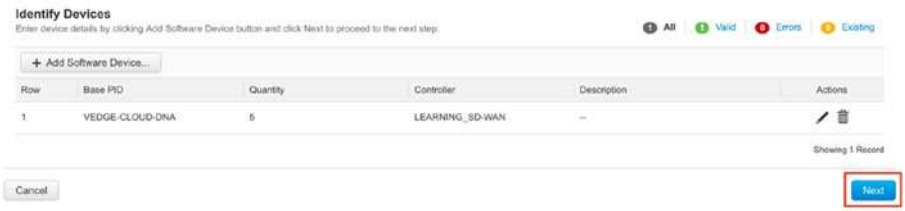

Figure 2-32. *Setting the number of vEdge devices*

Click Submit (Figure 2-33).

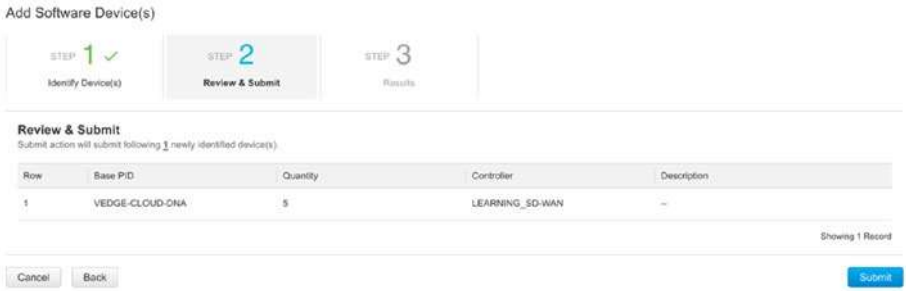

Figure 2-33. *Submitting the vEdge quantities*

Your request will be processed in the background, and you will receive an email once it has completed (I have obscured my work email address!) (Figure 2-34).

Figure 2-34. *Email confirmation*

Usually, this process is pretty quick, so give a few moments and refresh your page, and you should see your devices (Figure 2-35).

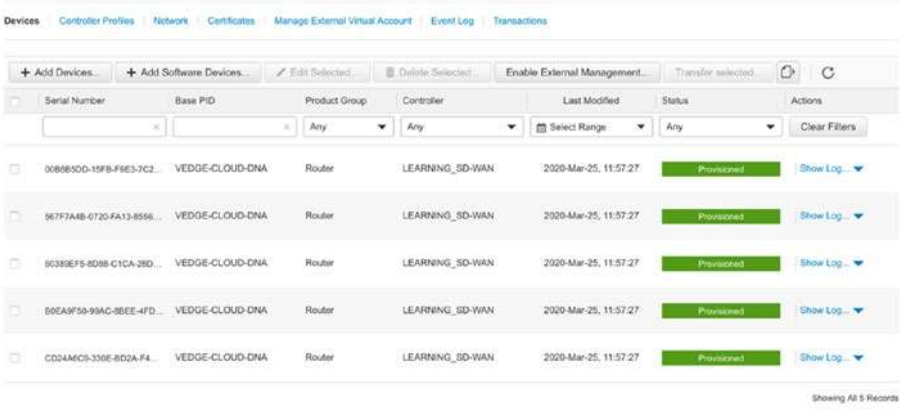

Figure 2-35. *Our device list*

Repeat the process, and add some CSR routers (Figure 2-36).

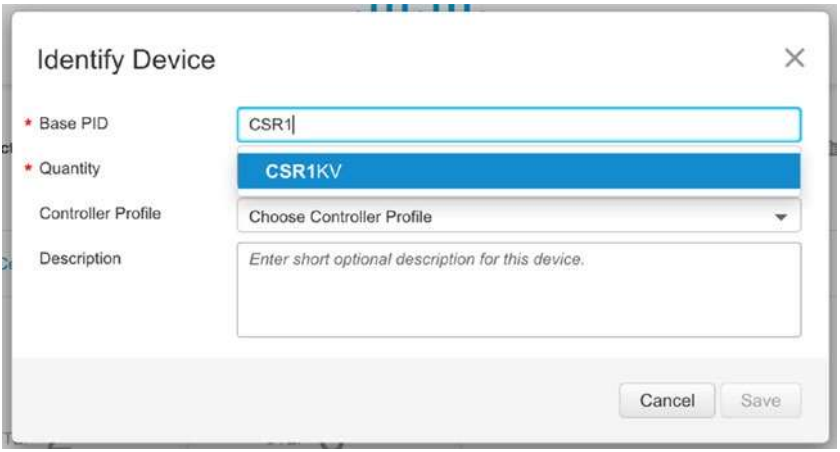

Figure 2-36. *Adding CSR devices*

Set the quantity and your profile (Figure 2-37).

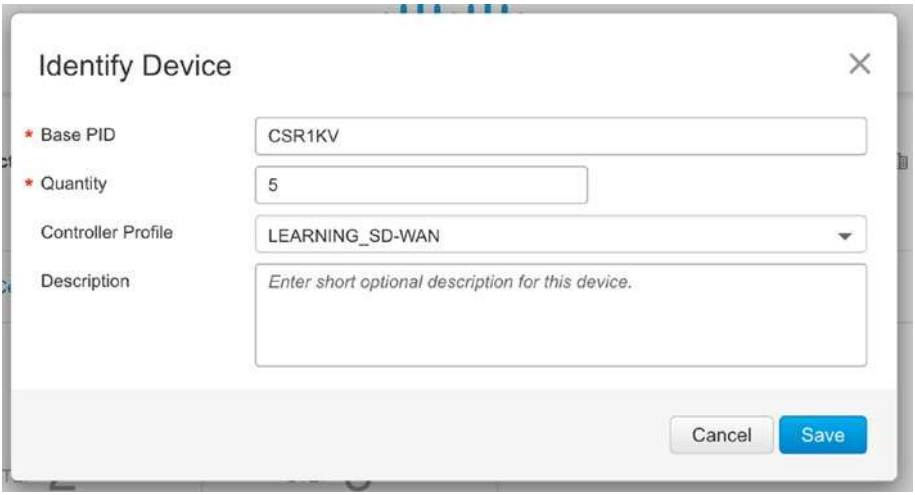

Figure 2-37. *5 CSR devices*

These should appear alongside the VEDGE devices (Figure 2-38).

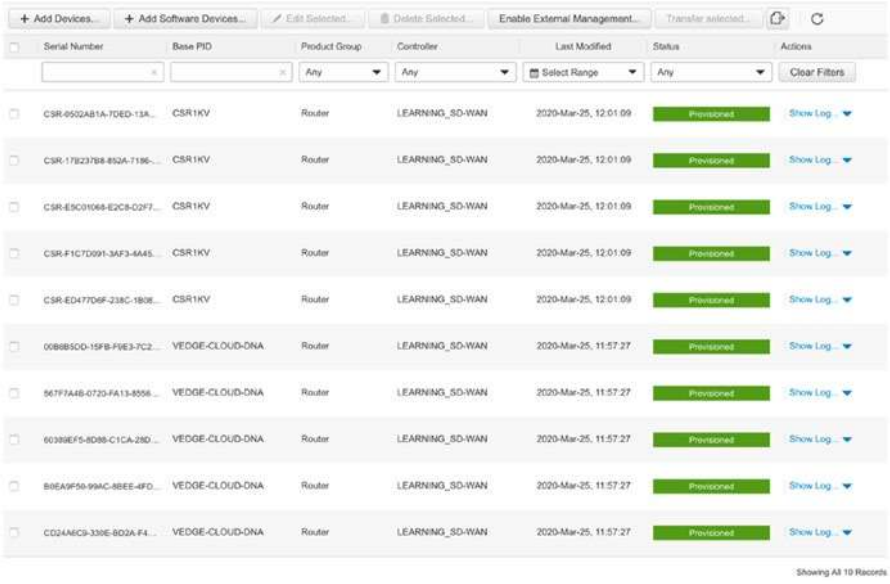

Figure 2-38. *Our device list in full*

Go back to Controller Profiles, and select the provisioning file (Figure 2-39).

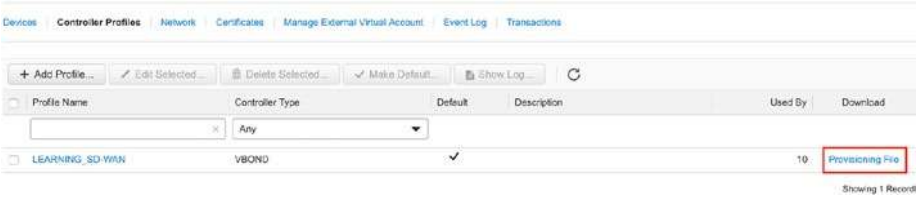

Figure 2-39. *Getting the provisioning file*

Select the "18.3 and newer" option, and download the file (Figure 2-40).

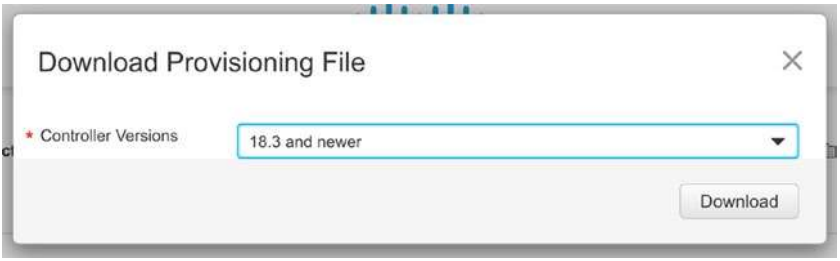

Figure 2-40. *Downloading the provisioning file*

The next step is to download the software. Because we are using EVE-NG, it is best to find the QCOW versions, so head back to the main Cisco website and search for "viptela qcow2" and download the vEdge, vSmart, vManage images. Also, download the SDWAN CSR1000v image (Figure 2-41).

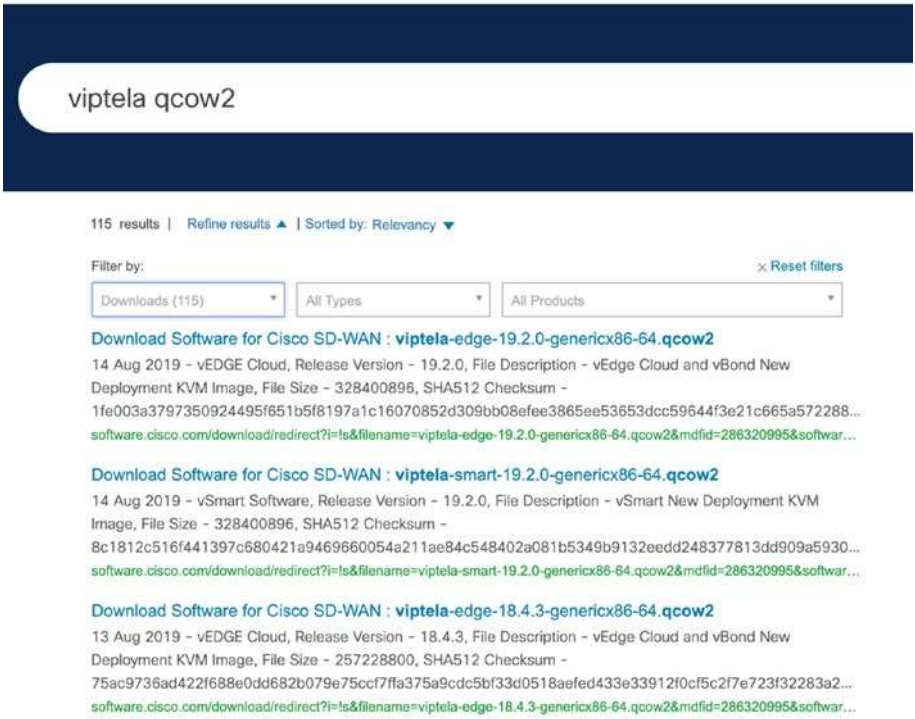

Figure 2-41. *The downloads we need*

The versions used in the majority of the book are either 19.2.1 or 19.3.0 for the Viptela images and 16.12.2r for the CSR1000v.

Note We do go through some upgrades later in the book, bumping all the "v" devices up to 20.1.1 and the CSR device up to 17.2.1r. The reasons for this is that firstly, upgrading is good practice, and secondly, the later versions include some features we need in the book, so the choice is yours as to whether to go straight for the later versions or follow along with the chapter on upgrades.

For the routers and switches, I am using vIOS 15.6 and vIOS-L2 15.2, respectively. Follow the steps on the EVE-NG website for how to generate these images from the VIRL originals: *www.eve-ng.net/index.php/documentation/howtos/howto-add-cisco-vios-from-virl/*. You can use other router and switch images (such as Cumulus VX) should you so wish.

The Ubuntu image is the desktop version, 17.10.1. You need to have OpenSSL installed. You can either create your own or download a ready-built image from here: *www.eve-ng.net/index.php/documentation/howtos/howto-create-own-linux-host-image/*.

If you are using different images, then please update the folder and file names accordingly.

Once you have downloaded the files, SSH onto your EVE-NG install and create the folders we will need:

```
cd /opt/unetlab/addons/qemu

mkdir vtbond-19.3.0
mkdir vtedge-19.3.0
mkdir vtsmart-19.3.0
mkdir vtmgmt19.3.0
mkdir csr1000vng-ucmk9.16.12.2r-sdwan
```

Then using FileZilla or WinSCP, copy the files as in Table 2-2.

Table 2-2. *Appliance versions*

Folder	Image
vtbond-19.3.0	viptela-edge-19.3.0-genericx86-64.qcow2
vtedge-19.3.0	viptela-edge-19.3.0-genericx86-64.qcow2
vtsmart-19.3.0	viptela-smart-19.3.0-genericx86-64.qcow2
vtmgmt-19.3.0	viptela-vmanage-19.3.0-genericx86-64.qcow2
csr1000vng-ucmk9.16.12.2r-sdwan	csr1000v-ucmk9.16.12.2r.qcow2

The next step is to rename the files and create a second disk for vManage:

```
cd csr1000vng-ucmk9.16.12.2r-sdwan
mv csr1000v-ucmk9.16.12.2r.qcow2 virtioa.qcow2
cd ..
cd vtbond-19.3.0/
mv viptela-edge-19.3.0-genericx86-64.qcow2 hda.qcow2
cd ..
cd vtedge-19.3.0/
mv viptela-edge-19.3.0-genericx86-64.qcow2 hda.qcow2
cd ..
cd vtsmart-19.3.0/
mv viptela-smart-19.3.0-genericx86-64.qcow2 hda.qcow2
cd ..
cd vtmgmt-19.3.0/
mv viptela-vmanage-19.3.0-genericx86-64.qcow2 hda.qcow2
/opt/qemu/bin/qemu-img create -f qcow2 hdb.qcow2 100G
cd ..
```

Lastly, update the wrappers using the command "`unl_wrapper -a fixpermissions`" (which corrects any permission issues after copying images around):

```
root@eve-ng:/opt/unetlab/addons/qemu# /opt/unetlab/wrappers/
unl_wrapper -a fixpermissions

root@eve-ng:/opt/unetlab/addons/qemu#
```

The resulting directory listing (with the router, switch, and Linux images) should look like this:

```
root@eve-ng:/opt/unetlab/addons/qemu# tree
.
├── csr1000vng-ucmk9.16.12.2r-sdwan
│   └── virtioa.qcow2
├── linux-ubuntu-desktop-17.10.1
│   └── virtioa.qcow2
├── vios-156
│   └── virtioa.qcow2
├── viosl2-152
│   └── virtioa.qcow2
├── vtbond-19.3.0
│   └── hda.qcow2
├── vtedge-19.3.0
│   └── hda.qcow2
├── vtmgmt-19.3.0
│   ├── hda.qcow2
│   └── hdb.qcow2
└── vtsmart-19.3.0
    └── hda.qcow2

8 directories, 9 files
root@eve-ng:/opt/unetlab/addons/qemu#
```

Topology

Our initial topology looks like this (Figure 2-42).

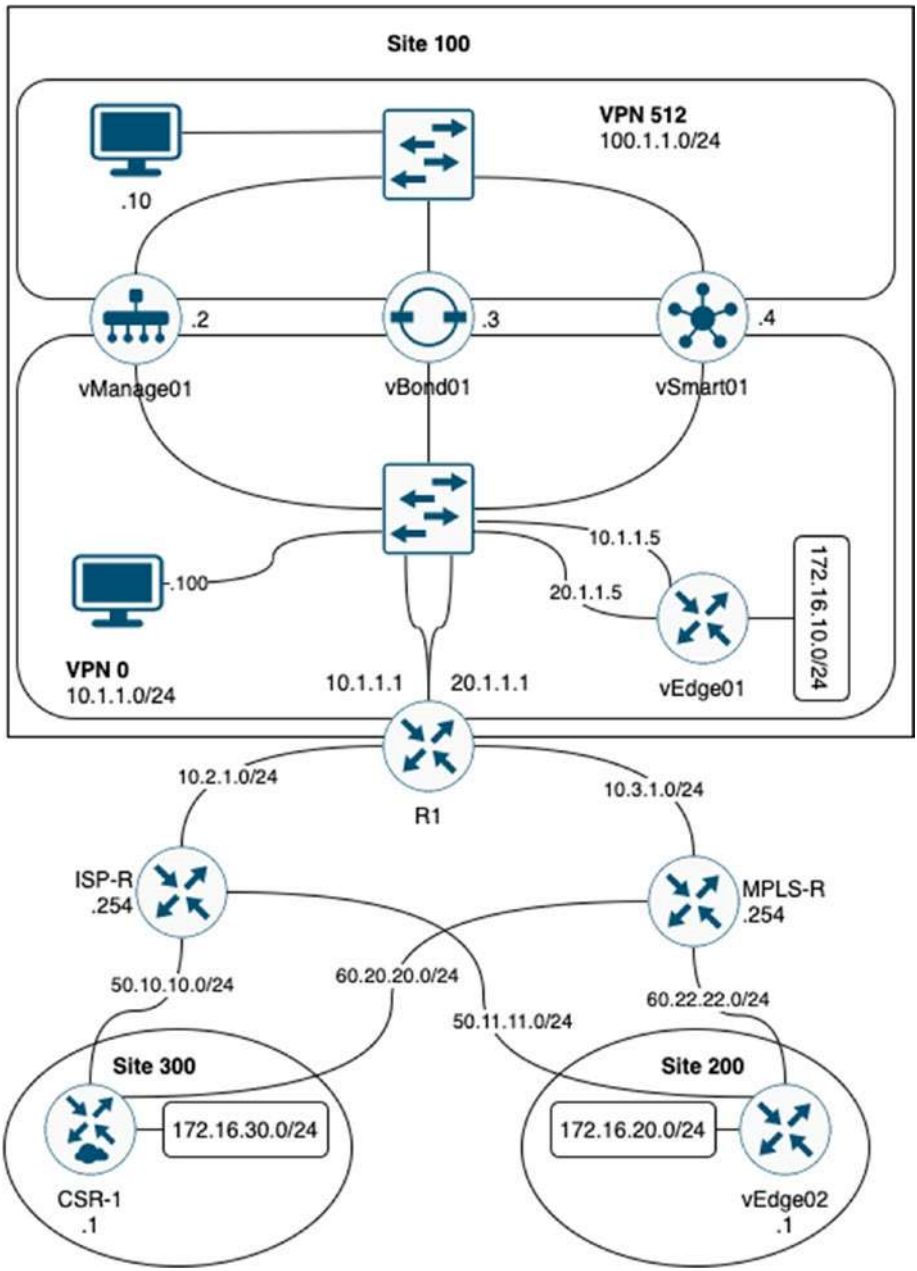

Figure 2-42. *Our initial topology*

We have a central site, our headquarters, which is designated as "site 100." We also have two branch sites, site 200 and site 300. These are connected to our HQ using MPLS as well as standard Internet links.

We have three networks that we are going to use SD-WAN to provide connectivity for, and these are 172.16.10.0/24 in site 100, 172.16.20.0/24 in site 200, and 172.16.30.0/24 in site 300.

We have Linux machines connected to VPN 512 (for management) and also to VPN 0, which we will mainly be using as our certificate server; this is the one we will be using most. If resources are an issue, then you can just use the Linux server in VPN 0.

The topology will change as we move through this book.

You can download the lab file from www.apress.com.

Importing the Lab File

To import the lab file, log into the EVE-NG GUI via your browser, and click the Import button. Browse to the zip file containing the lab and select it. Click the Upload button. You can then start the lab.

Initial Configurations

The initial configurations for R1, ISP-R, and MPLS-R are as follows. They have been truncated to just show the important configurations.

R1

```
R1#sh run
!
hostname R1
!
interface GigabitEthernet0/0
```

```
 description VPN0 Inside
 ip address 10.1.1.1 255.255.255.0
 no shut
!
interface GigabitEthernet0/1
 description VPN0 Outside
 ip address 10.2.1.1 255.255.255.0
 no shut
!
interface GigabitEthernet0/2
 description MPLS Outside
 ip address 10.3.1.1 255.255.255.0
 no shut
!
interface GigabitEthernet0/3
 description MPLS Inside
 ip address 20.1.1.1 255.255.255.0
!
ip route 50.0.0.0 255.0.0.0 10.2.1.254
ip route 60.0.0.0 255.0.0.0 10.3.1.254
!
end

R1#
```

ISP-R

```
ISP-R#sh run
!
hostname ISP-R
!
interface GigabitEthernet0/0
```

```
 description VPN0
 ip address 10.2.1.254 255.255.255.0
 no shut
!
interface GigabitEthernet0/1
 Link to vEdge02
 ip address 50.11.11.254 255.255.255.0
 no shut
!
interface GigabitEthernet0/2
 no ip address
!
interface GigabitEthernet0/3
 Link to CSR-1
 ip address 50.10.10.254 255.255.255.0
 no shut
!
ip route 10.1.1.0 255.255.255.0 10.2.1.1
!
end

ISP-R#
```

MPLS-R

```
MPLS-R#sh run
!
hostname MPLS-R
!
interface GigabitEthernet0/0
 description MPLS
 ip address 10.3.1.254 255.255.255.0
```

```
 no shut
!
interface GigabitEthernet0/1
 Link to vEdge02
 ip address 60.22.22.254 255.255.255.0
 no shut
!
interface GigabitEthernet0/2
 Link to CSR-1
 ip address 60.20.20.254 255.255.255.0
 no shut
!
ip route 10.1.1.0 255.255.255.0 10.3.1.1
!
end

MPLS-R#
```

ESXi and KVM Configuration

This network can also be set up on VMWare and KVM. While the steps for creating each of the SD-WAN VMs will be given for both platforms at the end of each respective chapter, explaining the full set up of ESXi, vCenter, and KVM is beyond the scope of this book.

Summary

We have set up our EVE-NG server, created our smart account, generated our serial file list so that we can run some edge devices, and downloaded our device images. We should also have the routers configured, ready to start sending traffic around our network. With that in mind, let's get started with our vManage server.

CHAPTER 3

Deploying vManage

In this chapter, we will set up our vManage servers. vManage is the NMS (Network Management System) which controls our SD-WAN, so it makes sense that we start here. We will set up our organization, the certificates we need to add and authenticate our devices, and look at how to control our users, implementing clustering for high availability and resilience. We will also look at single- and multi-tenancy options, installation of vManage on ESXi and KVM, and how to install the serial file we generated from the Smart account.

Installing vManage

Because we are using a ready-made appliance, all the heavy lifting has been done for us, so just right-click vManage01 in EVE-NG and select "Start," and then left-click it to start the telnet console session.

After a while, you will be prompted to log in; use the default username and password of "admin", setting a new password when prompted. The next step is to format the storage disk, so select the 100G disk created earlier, and press "y" to format it. Once this is done, the system will reboot.

```
vmanage login: admin
Password:
Welcome to Viptela CLI
admin connected from 127.0.0.1 using console on vmanage
```

```
Available storage devices:
hdb     100GB
hdc     3GB
1) hdb
2) hdc
Select storage device to use: 1
Would you like to format hdb? (y/n): y
mke2fs 1.43.8 (1-Jan-2018)
Creating filesystem with 26214400 4k blocks and 6553600 inodes
Filesystem UUID: bc995938-6aeb-49c9-aeba-48d70b18235b
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736,
        1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done
Writing inode tables: done
Creating journal (131072 blocks): done
Writing superblocks and filesystem accounting information: done

vmanage# wall: cannot get tty name: Success

Broadcast message from root@vmanage (somewhere) (Mon Mar 30
09:35:19 2020):

Mon Mar 30 09:35:19 UTC 2020: The system is going down for
reboot NOW!

Stopping services...
acpid: exiting
```

When the system has come back up again, log in and set the system IP address (100.100.1.2), the site-id (100), the organization name (Learning_SD-WAN), and the hostname (vManage01) (as shown in Figure 2-42 in the previous chapter).

```
Mon Mar 30 09:36:19 UTC 2020: System Ready

viptela 19.3.0

vmanage login: admin
Password:
Welcome to Viptela CLI
admin connected from 127.0.0.1 using console on vmanage
vmanage#
vmanage#
vmanage# config
Entering configuration mode terminal
vmanage(config)# system
vmanage(config-system)# system-ip 100.100.1.2
vmanage(config-system)# site-id 100
vmanage(config-system)# organization-name Learning_SD-WAN
vmanage(config-system)#
vmanage(config-system)# host-name vManage01
vmanage(config-system)#
```

The next step is to create the management VPN (VPN 512).

```
vmanage(config-system)# vpn 512
vmanage(config-vpn-512)# interface eth0
vmanage(config-interface-eth0)# ip address 100.1.1.2/24
vmanage(config-interface-eth0)# no shutdown
vmanage(config-interface-eth0)#
vmanage(config-interface-eth0)# exit
vmanage(config-vpn-512)#
```

The management VPN carries the out-of-band network management traffic for the SD-WAN devices.

Now we can create our WAN overlay VPN (VPN 0):

```
vmanage(config-vpn-512)# vpn 0
vmanage(config-vpn-0)# no interface eth0
vmanage(config-vpn-0)# interface eth1
vmanage(config-interface-eth1)# ip address 10.1.1.2/24
vmanage(config-interface-eth1)# tunnel-interface
vmanage(config-tunnel-interface)# exit
vmanage(config-interface-eth1)# no shutdown
vmanage(config-interface-eth1)#
vmanage(config-interface-eth1)# exit
vmanage(config-vpn-0)#ip route 0.0.0.0/0 10.1.1.1
vmanage(config-vpn-0)# exit
vmanage(config)#
```

VPN 0 carries our control traffic. By default, all of our interfaces will be added to VPN 0, and the interfaces are disabled. We, therefore, have to remove eth0 from VPN 0 (because we need eth0 to be in VPN 512) and enable eth1, configuring the IP address and setting it as a tunnel interface. We also add a default route. You will notice that no default route was added to our VPN 512 configuration, but this is due to the size of our topology; in a real-world set up, this may be required.

The last step of our configuration is to save our settings. Before we try and commit our changes, we should confirm that the settings are valid, which we can do by using the "`validate`" command or "`commit check`".

```
vmanage(config)# validate
Validation complete
vmanage(config)# commit check
Validation complete
vmanage(config)# commit and-quit
Commit complete.
vManage01#
```

If all is working, we should have connectivity to the R1 router:

```
vManage01# ping 10.1.1.1
Ping in VPN 0
PING 10.1.1.1 (10.1.1.1) 56(84) bytes of data.
64 bytes from 10.1.1.1: icmp_seq=1 ttl=64 time=1.61 ms
64 bytes from 10.1.1.1: icmp_seq=2 ttl=64 time=0.825 ms
64 bytes from 10.1.1.1: icmp_seq=3 ttl=64 time=0.787 ms
64 bytes from 10.1.1.1: icmp_seq=4 ttl=64 time=0.680 ms
64 bytes from 10.1.1.1: icmp_seq=5 ttl=64 time=0.743 ms
^C
--- 10.1.1.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5001ms
rtt min/avg/max/mdev = 0.680/0.897/1.611/0.322 ms
vManage01#
```

From the Linux machine in VPN 0, launch the browser and connect to the management interface on `https://10.1.1.2` (Figure 3-1).

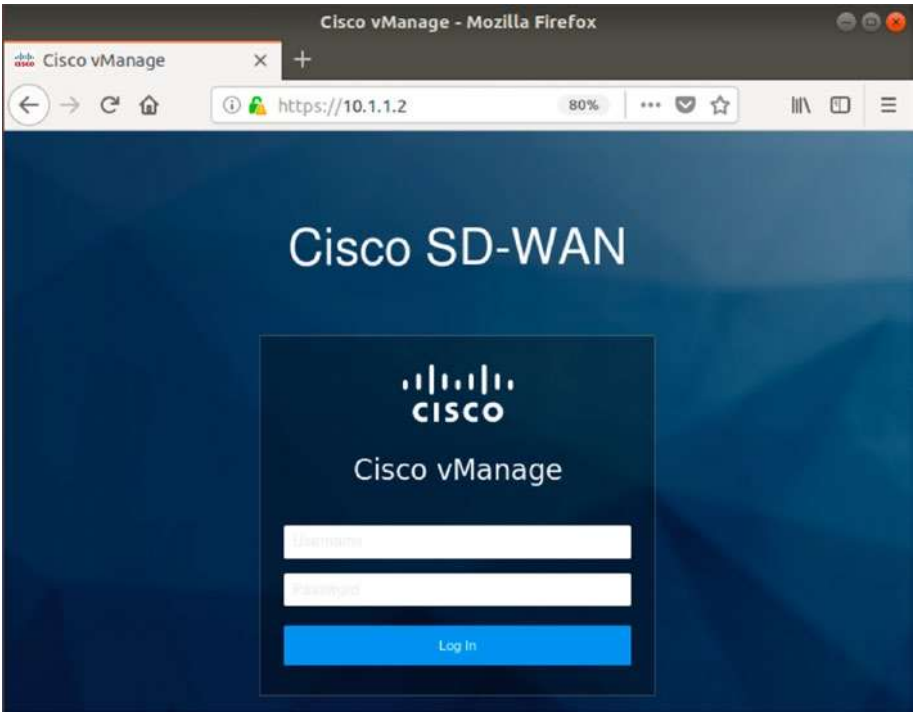

Figure 3-1. *The vManage login page*

You can also use the Linux machine in VPN 512 and connect using the URL `https://100.1.1.2`. Log in using the username admin and the password you set during setup.

TIP There is a lot of information on the dashboard, so if you find your screen resolution is too small, as it can be with QEMU devices, then you can change it if you set your QEMU VM properties to use "`-vga virtio`" instead of "`-vga std`", as shown in Figure 3-2. You will need to stop and then start the Linux VM for the change to take effect.

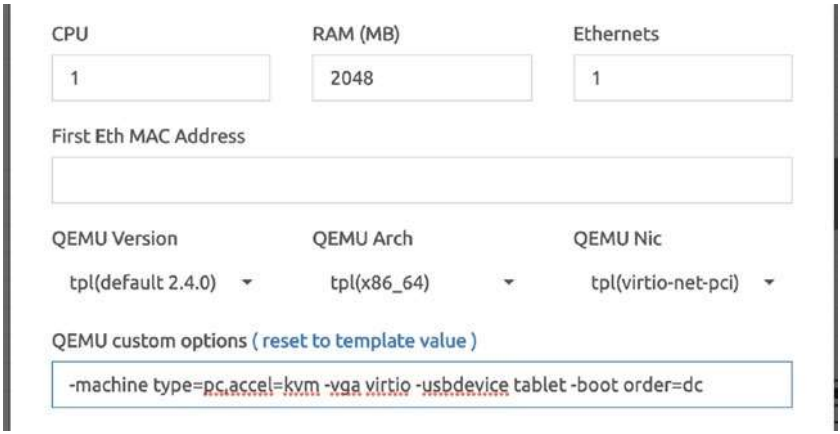

Figure 3-2. *Use -vga virtio for a bigger window size*

The vManage interface (Figure 3-3) shows us our connected devices at the top, listing the vSmart, WAN Edge, vBond devices, and vManage servers.

The middle sections show us our WAN status and site health, and at the bottom, we have our applications and application-aware routing.

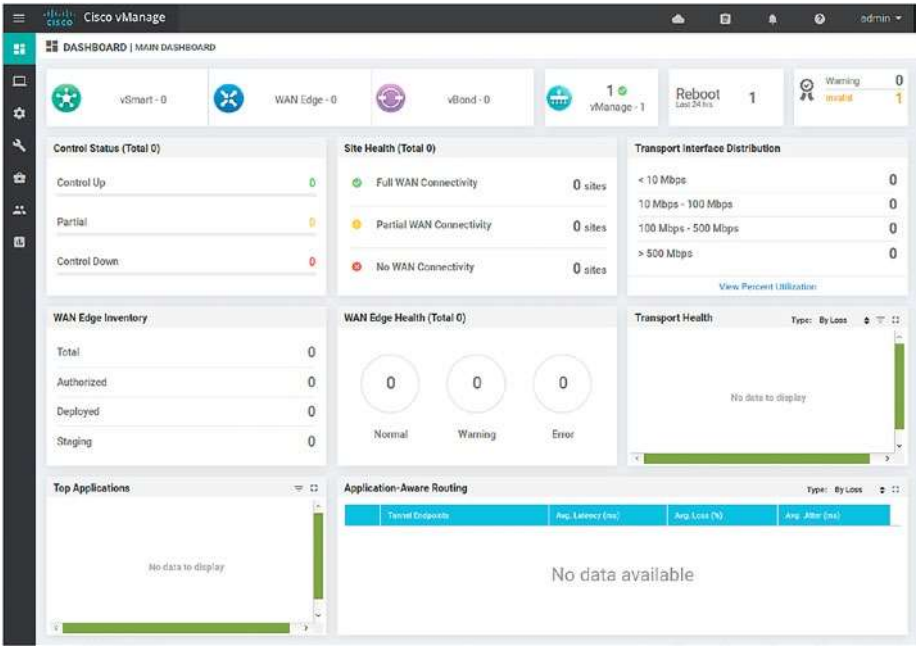

Figure 3-3. *The main vManage dashboard*

There is an issue at the moment, though. We can tell this as there is a (invalid) warning at the top right-hand corner (Figure 3-4).

Figure 3-4. *Certificate invalid message*

Clicking the number takes us into the error detail, where we can see that we have a problem with our certificate (Figure 3-5).

Figure 3-5. *Certificate issue detail*

Certificates

Certificates in the SD-WAN network are hugely important; they form a very large role in the authentication of devices.

We need to generate the CA certificates and upload them to the vManage server. The easiest way to do this is to enable SSH on the vManage server so that we can use SCP (Secure Copy Protocol) to upload the files we need. We enable this under the VPN 0 tunnel interface.

```
vManage01(config)# vpn 0
vManage01(config-vpn-0)# interface eth1
vManage01(config-interface-eth1)# tunnel-interface
vManage01(config-tunnel-interface)# allow-service sshd
vManage01(config-tunnel-interface)#
vManage01(config-tunnel-interface)# end
Uncommitted changes found, commit them? [yes/no/CANCEL] yes
Commit complete.
vManage01#
```

We will be using the Linux host to be our certificate authority (CA) so that it can sign our certificates.

The first step is to set it up as our CA. Open the console window on the Linux host, and enter the following:

```
openssl genrsa -out CA.key 2048
openssl req -new -x509 -days 1000 -key CA.key -out CA.crt
```

As mentioned in the previous chapter, your Linux VM will need to have the OpenSSL packages installed for this to work. This will generate our CA certificate. Next, copy it across to the vManage server using SCP ("`scp CA.crt admin@10.1.1.2:`"), typing in the vManage admin password when prompted.

Switching back to the vManage server, install the CA certificate, using the command "`request root-cert-chain install`":

```
vManage01# request root-cert-chain install /home/admin/CA.crt
Uploading root-ca-cert-chain via VPN 0
Copying ... /home/admin/CA.crt via VPN 0
Updating the root certificate chain..
Successfully installed the root certificate chain
vManage01#
```

Now that the vManage server knows that we have a CA, we can use it to sign our CSRs.

We must have our organization name set at this stage. If it is not set, then go to *Administration* ➤ *Settings* and set it.

As we are running this in a virtual lab, we need to switch how our certificates are provisioned to us. Go to *Administration* ➤ *Settings*, and click "edit" next to "Controller Certificate Authority."

By default, it is set to "Cisco" as shown in Figure 3-6.

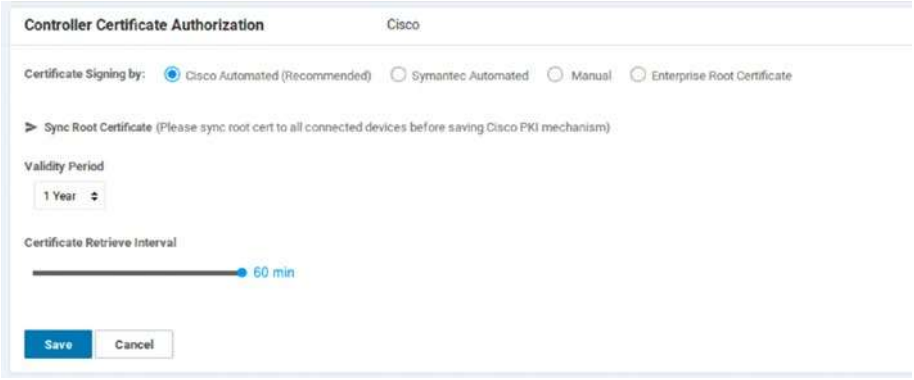

Figure 3-6. *Default certificate authority*

Click the "Enterprise Root Certificate" option (Figure 3-7), and confirm the change by clicking "Proceed."

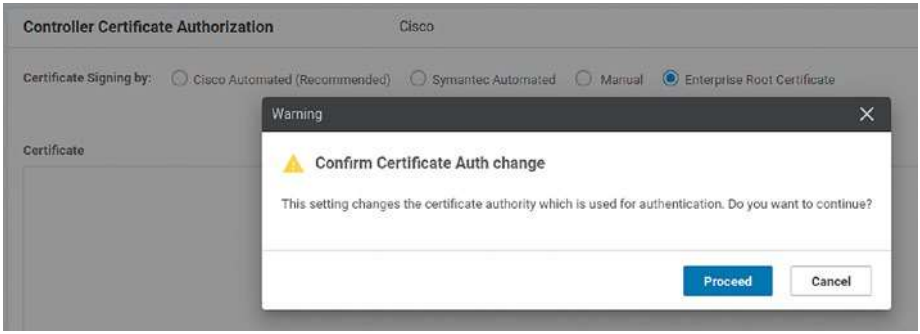

Figure 3-7. *Confirm the change*

Upload the CA.crt file next (Figure 3-8).

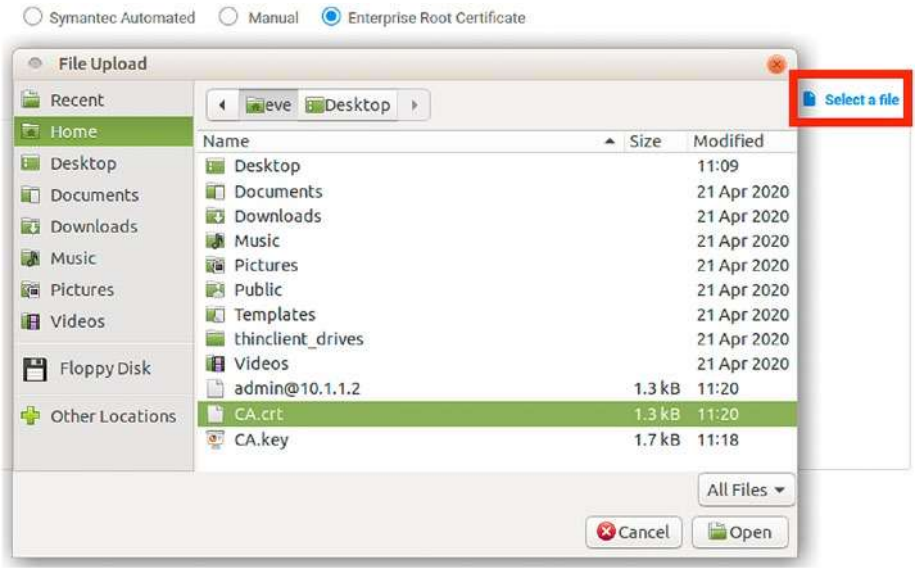

Figure 3-8. *Upload the CA certificate*

Click “Import & Save” (Figure 3-9).

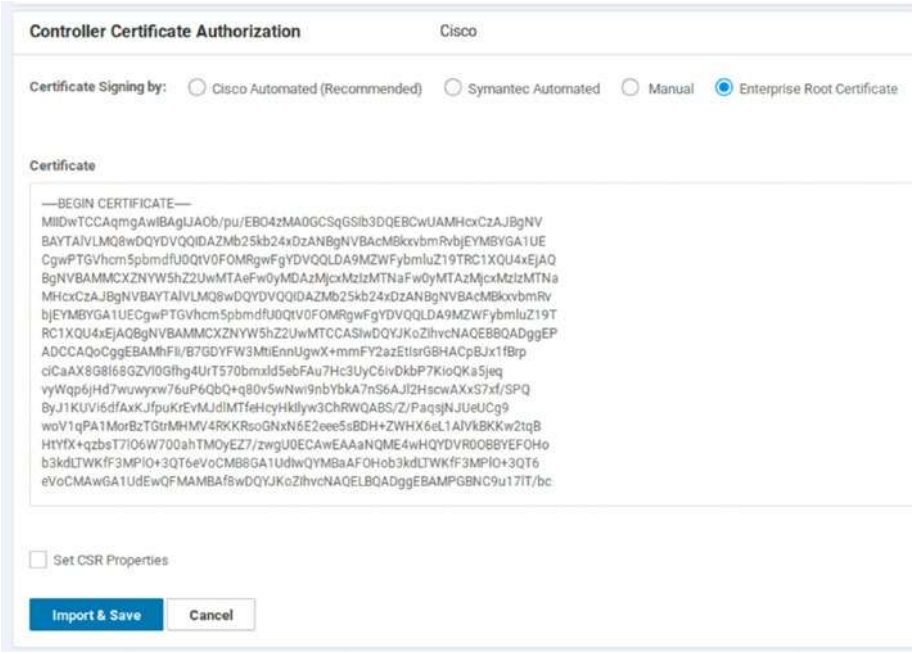

Figure 3-9. *Import and save the CA certificate*

Generate a certificate request by going to *Configuration* ➤ *Certificates* (Figure 3-10).

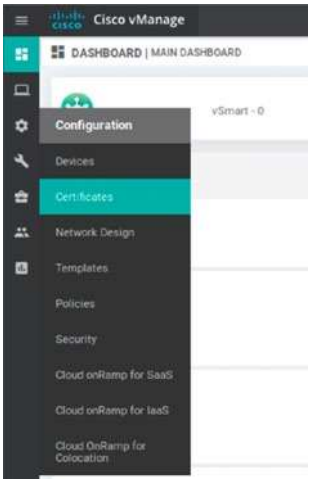

Figure 3-10. *The certificates menu*

There won't be any certificates for the WAN Edge yet (Figure 3-11), so click "Controllers."

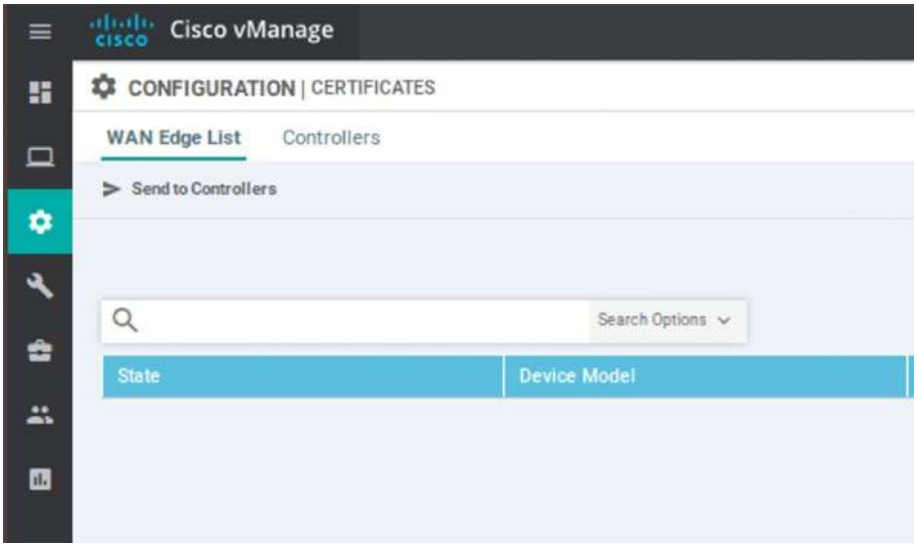

Figure 3-11. *Certificates (or lack of)*

The Controllers page will show "No certificate installed." Click the three buttons on the right-hand side, and select "Generate CSR"; vManage will generate a certificate (Figure 3-12).

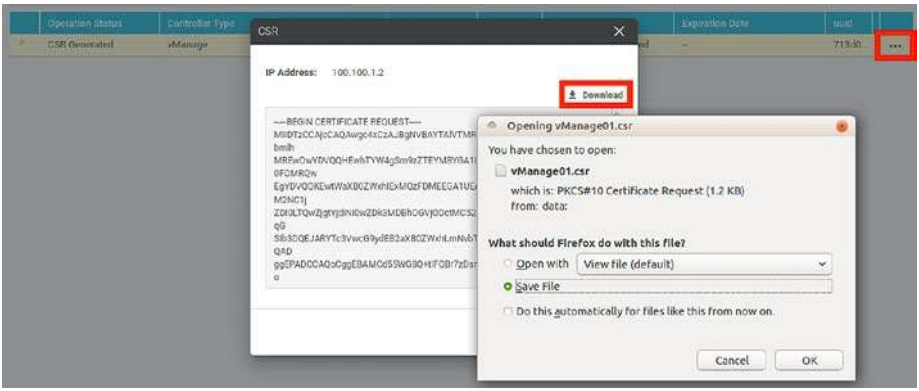

Figure 3-12. *vManage CSR*

Download the CSR and save it. Back on the Linux machine, sign the CSR, making sure that the CSR is in the right directory (Figure 3-13):

```
openssl x509 -req -in vManage01.csr -CA CA.crt -CAkey CA.key -
CAcreateserial -out vManage01.crt -days 1000 -sha256
```

```
user@user-virtual-machine:~/Downloads$ openssl x509 -req -in vManage01.csr -CA
CA.crt -CAkey CA.key \
> -CAcreateserial -out vManage01.crt -days 1000 -sha256
Signature ok
subject=/C=US/ST=California/L=San Jose/OU=Learning_SD-WAN/O=Viptela LLC/CN=vman
age-b63aa480-37f6-4589-ba5e-961c9115c8fe-1.viptela.com/emailAddress=support@vip
tela.com
Getting CA Private Key
user@user-virtual-machine:~/Downloads$
```

Figure 3-13. *Signing the vManage CSR*

Once the certificate has been generated, go back to the vManage GUI, and select the "Install Certificate" option. Click "Select a file," and select the vManage01.crt file we just created (Figure 3-14).

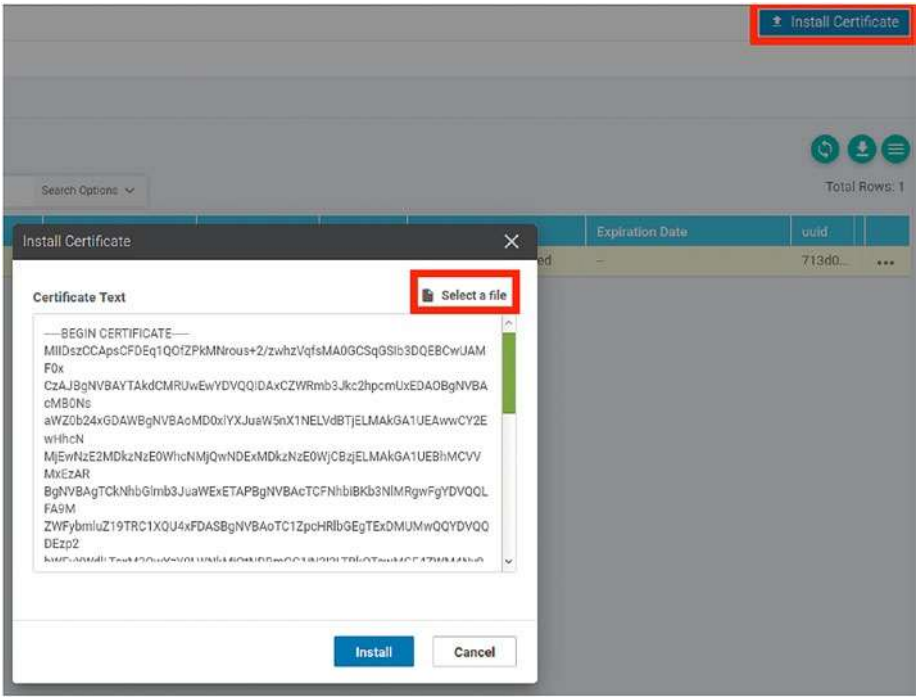

Figure 3-14. *Installing the signed vManage certificate*

Click Install. The certificate will then synchronize (Figure 3-15).

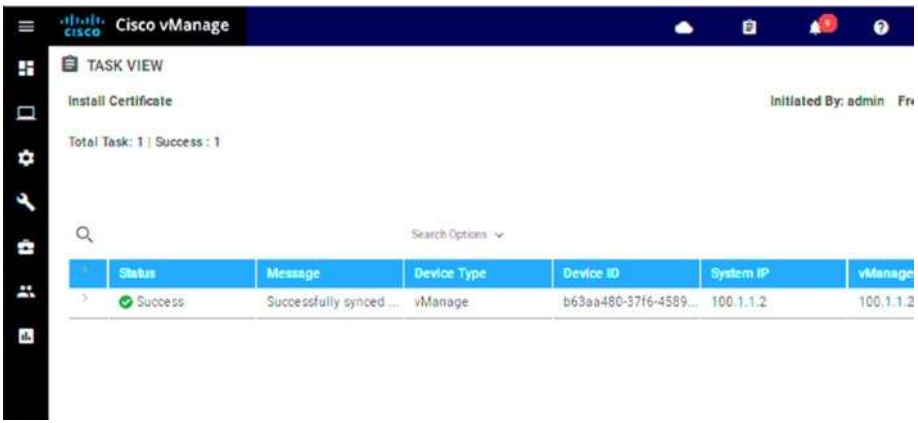

Figure 3-15. *Certificate synchronization*

If the certificate takes a while to synchronize, you can speed it up using the API, by going to `https://10.1.1.2/dataservice/system/device/sync/rootcertchain` (Figure 3-16).

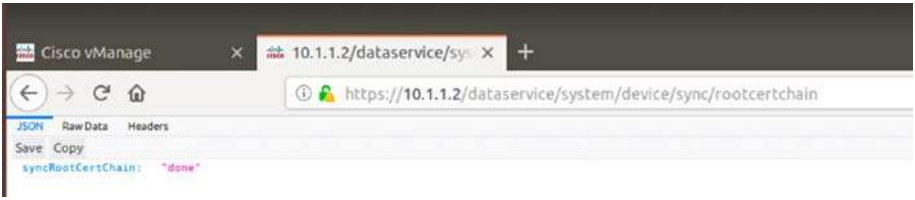

Figure 3-16. *Certificate synchronization through the API*

While our certificate synchronizes, let's take a moment to look at users and clustering.

Users

We only have one user at the moment, the admin user. We can create more, if we want to, by going to *Administration* ➤ *Manage Users*.

There are three groups (by default) that we add users into, and these are basic, netadmin, and operator. The NetAdmin role has full rights over every aspect of the SD-WAN software. The operator has read-only rights, and the basic role can only look at the interface and the system. The full list of privileges is listed in Table 3-1.

Table 3-1. *User privileges*

	Basic	NetAdmin	Operator
Alarms	–	Full	Read
Audit Log	–	Full	Read
Certificates	–	Full	Read
Cloud OnRamp	–	Full	Read
Cluster	–	Full	Read
Colocation	–	Full	Read
Device inventory	–	Full	Read
Device monitoring	–	Full	Read
Device reboot	–	Full	Read
Events	–	Full	Read
Interface	Read	Full	Read
Manage users	–	Full	Read
Policy	–	Full	Read
Policy configuration	–	Full	Read
Policy deploy	–	Full	Read
Routing	–	Full	Read
Security	–	Full	Read
Security policy configuration	–	Full	Read
Settings	–	Full	Read
Software upgrade	–	Full	Read

(*continued*)

Table 3-1. *(continued)*

	Basic	NetAdmin	Operator
System	Read	Full	Read
Template configuration	–	Full	Read
Template deploy	–	Full	Read
Tools	–	Full	Read
vAnalytics	–	Full	Read

We can create custom groups by clicking the "Add User Group" button, and we can also create new users, by clicking the "Add User" button. Here, we can create an account for "monitoring," which is a member of the "basic" user group (Figure 3-17).

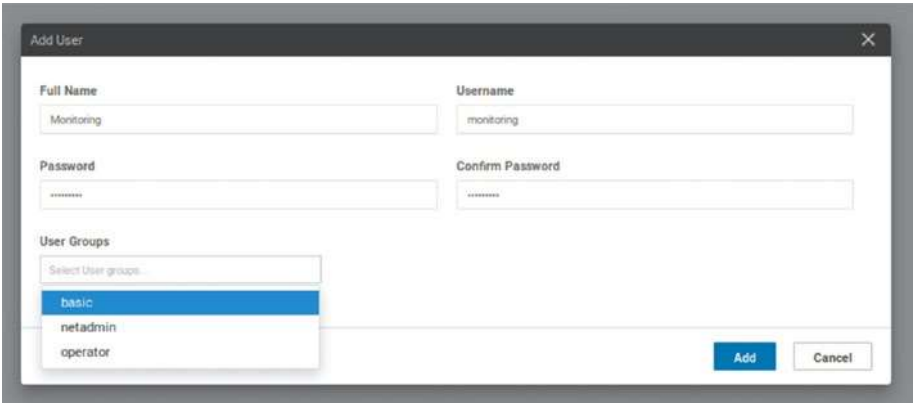

Figure 3-17. *A basic user*

Our new user is visible along with the default admin account (Figure 3-18).

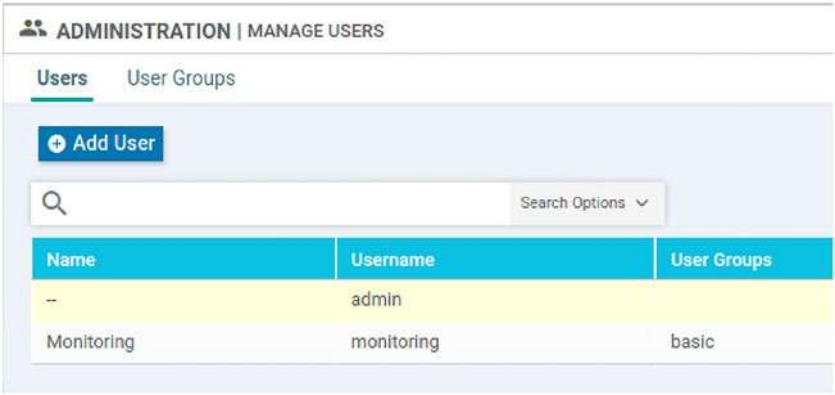

Figure 3-18. *A new user*

vManage Clustering

As vManage is the linchpin of the SD-WAN, it needs to be resilient. We can have multiple vManage servers and cluster them to ensure the configuration among them is consistent.

Navigate to *Administration* ➤ *Cluster Management* (Figure 3-19).

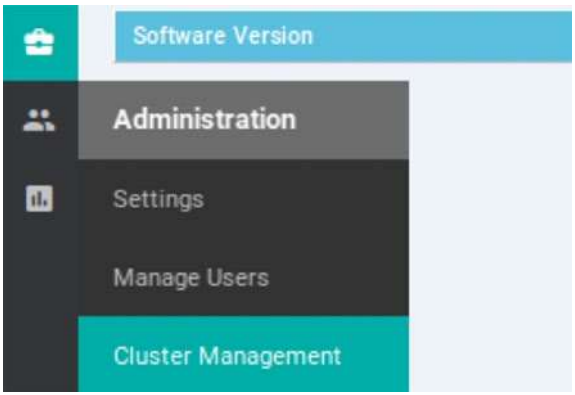

Figure 3-19. *Cluster Management*

By default, the local vManage server will show an IP address of "localhost" as shown in Figure 3-20.

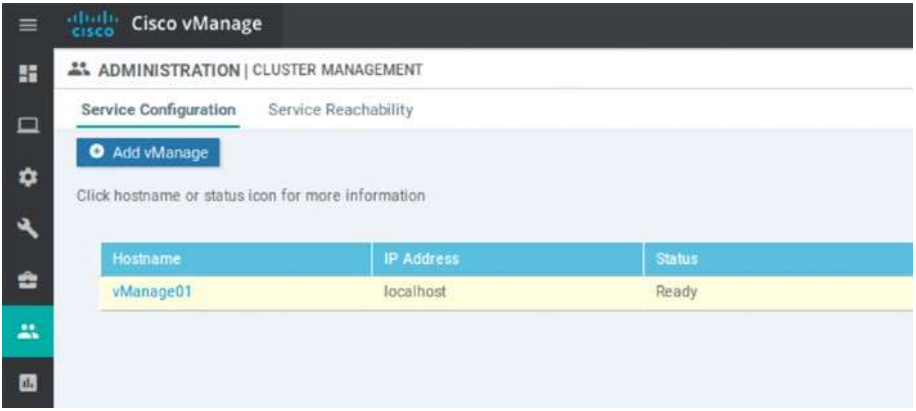

Figure 3-20. *vManage as localhost*

We need to be able to distinguish between our vManage servers, so we have to change the IP address first, before we can add any new servers to our cluster.

Click the triple dots at the right-hand side of the server, and select "Edit" (Figure 3-21).

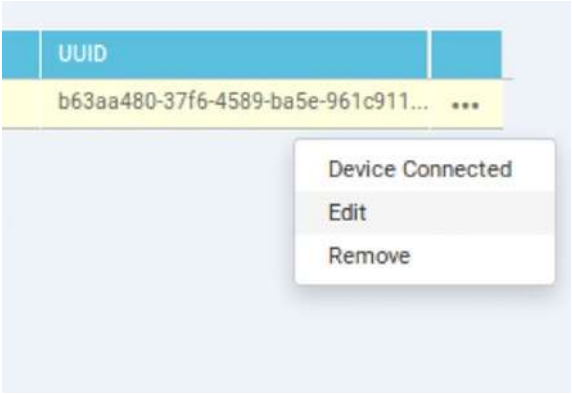

Figure 3-21. *Editing the vManage server*

Select the IP address (10.1.1.2) from the drop-down, and enter the username and password (Figure 3-22).

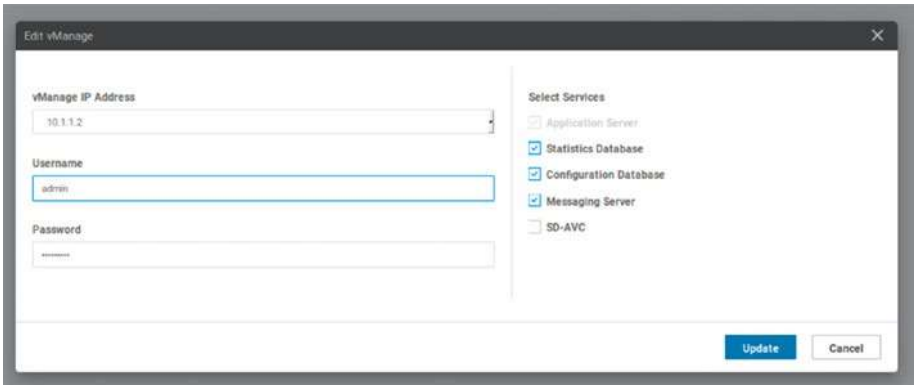

Figure 3-22. *Setting the vManage IP address*

Click "Update." In the next window, you will be prompted to reboot the vManage server (Figure 3-23).

Figure 3-23. *Rebooting the vManage*

Click OK. You will see now that the IP address has changed (Figure 3-24).

Figure 3-24. *New vManage IP address*

We can now add a new vManage node, and we can do this by copying the existing one on the EVE-NG server:

```
root@eve-ng:~# cd /opt/
root@eve-ng:/opt# cd unetlab/addons/qemu/
root@eve-ng:/opt/unetlab/addons/qemu# cp -R vtmgmt-19.3.0
vtmgmt-19.3.0-2
root@eve-ng:/opt/unetlab/addons/qemu# /opt/unetlab/wrappers/
unl_wrapper -a fixpermissions

root@eve-ng:/opt/unetlab/addons/qemu#
```

Add a new vManage server to our topology by right-clicking on a blank area of the topology in EVE-NG and selecting "Node" (Figure 3-25).

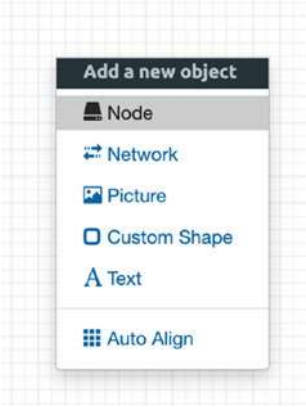

Figure 3-25. *Adding a new node to EVE-NG*

Scroll down until you see "Viptela vManage," and click it (Figure 3-26).

Figure 3-26. *Adding the second vManage server*

Select the second instance we just created (Figure 3-27).

Figure 3-27. *Select the second vManage node*

Name it "vManage02" and click Save.

Hover over the new instance with your mouse and then drag the orange network cable icon over to Net100, making sure that eth0 is connected to Net100. You will need to right-click on the VPN0 switch and stop it, then connect the eth1 interface from vManage02 to Gi1/0 on the switch. Start the switch again.

Right-click the topology, and add a text object of ".22". The new topology should look like Figure 3-28.

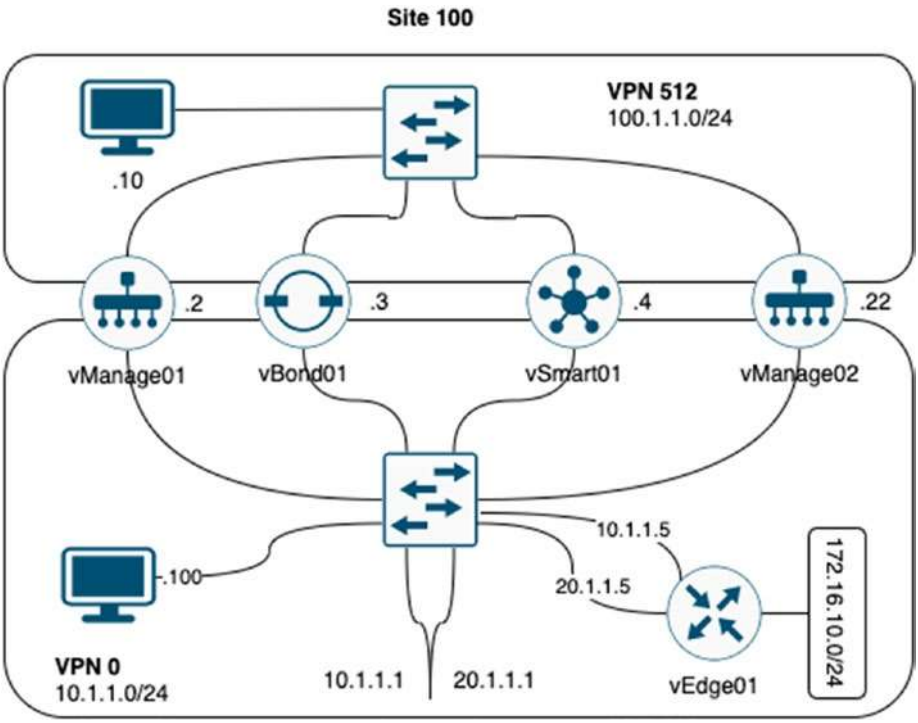

Figure 3-28. *The new topology*

Right-click vManage02 and select start (Figure 3-29).

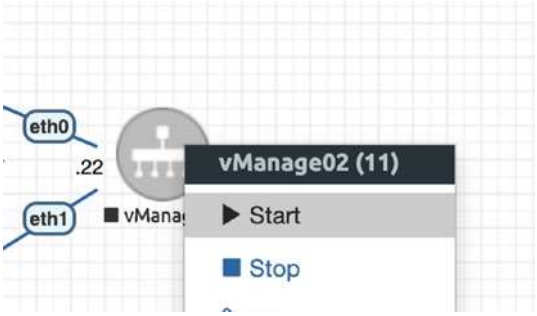

Figure 3-29. *Starting vManage02*

Left-click it to launch a telnet connection.

Once the console has loaded, you will be prompted to change the password and format the disk, as we did at the start of the chapter. Once the VM has rebooted, configure it as follows:

```
vmanage# config
Entering configuration mode terminal
vmanage(config)# system
vmanage(config-system)# system-ip 100.100.1.22
vmanage(config-system)# site-id 100
vmanage(config-system)# organization-name Learning_SD-WAN
vmanage(config-system)# host-name vManage02
vmanage(config-system)#
vmanage(config-system)# vpn 512
vmanage(config-vpn-512)# interface eth0
vmanage(config-interface-eth0)# ip address 100.1.1.22/24
vmanage(config-interface-eth0)# no shutdown
vmanage(config-interface-eth0)# exit
vmanage(config-vpn-512)#
vmanage(config-vpn-512)# vpn 0
vmanage(config-vpn-0)# no interface eth0
vmanage(config-vpn-0)# interface eth1
vmanage(config-interface-eth1)# ip address 10.1.1.22/24
vmanage(config-interface-eth1)# no shut
vmanage(config-interface-eth1)# tunnel-interface
vmanage(config-tunnel-interface)# allow-service all
vmanage(config-tunnel-interface)# exit
vmanage(config-interface-eth1)# ip route 0.0.0.0/0 10.1.1.1
vmanage(config-vpn-0)# end
Uncommitted changes found, commit them? [yes/no/CANCEL] yes
Commit complete.
vManage02# ping 10.1.1.2
```

```
Ping in VPN 0
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.623 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.647 ms
^C
--- 10.1.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.623/0.635/0.647/0.012 ms
vManage02#
```

I have been a bit lazy with the preceding settings and allowed all services, but this does make life a lot easier when setting up a cluster. In production, I would say that as a minimum, SSHD and NetConf should be allowed, but each deployment will have its separate requirements.

Before we can add this new vManage server to the cluster, we need to complete the setup, which means we need to follow the same steps as we did with vManage01 and add the enterprise root certificates and create a certificate for vManage02.

Once this is complete, we should not have any warnings on the main dashboard on vManage02.

Repeat the process of setting the IP address under *Administration* ➤ *Cluster Management* as we did for vManage01, and wait for vManage02 to come back up again.

Returning to vManage01, we can now add vManage02 to the cluster, click the "Add vManage" button, and enter the IP address of vManage02 and the username and password (Figure 3-30).

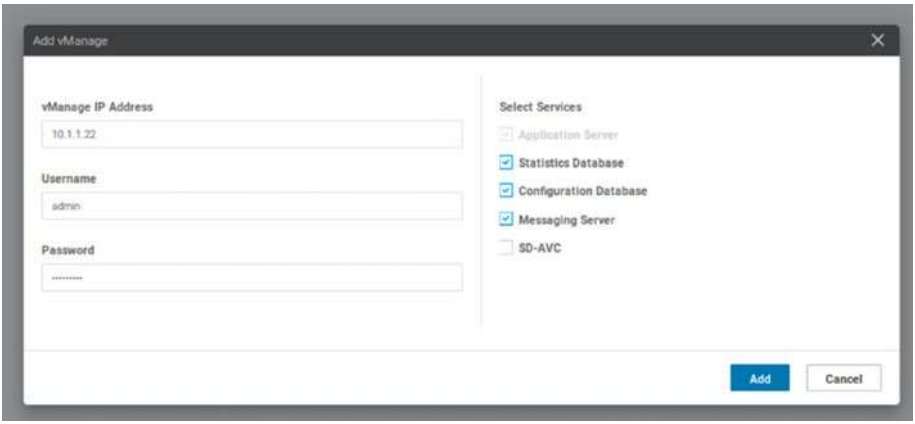

Figure 3-30. *Adding vManage02 to the cluster*

Click "Add" and accept the prompt to reboot vManage02 (Figure 3-31).

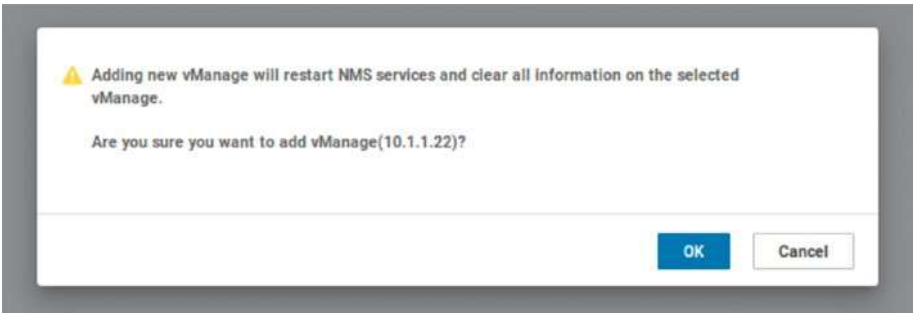

Figure 3-31. *Rebooting vManage02*

The new server will have a state of pending until it has rebooted (Figure 3-32).

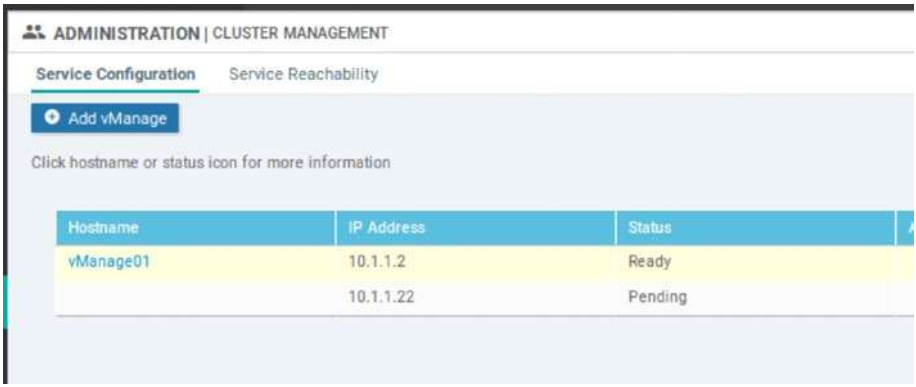

Figure 3-32. *vManage02 is pending*

Once the server has rebooted, it should be part of the cluster. This will be reflected in the dashboard, as the number of vManage systems should now be 2. We can also see this in the cluster details, as the hostname has changed and the status is now "Ready" (Figure 3-33).

Figure 3-33. *Both vManage servers are ready*

Single- and Multi-tenancy Options

Cisco's SD-WAN has tenancy options, so we can enable multi-tenancy support; however, once we enable multi-tenancy, we cannot go back to single-tenancy mode, so don't go through these steps (unless you really fancy rebuilding your vManage servers)!

To enable multi-tenancy, go to *Administration* ➤ *Settings*. Select the Edit button next to "Tenancy Mode," and select "Multitenant," enter a domain name and a cluster-ID, and then click Save (Figure 3-34).

Figure 3-34. *Setting up tenancies*

The vManage server will then reboot and come up in multi-tenancy mode.

To create tenants, go to the new vManage URL, which, in this example, would be `https://1.acme.org`. From the main screen, go to *Administration* ➤ *Tenant Management* and click "Add Tenant." Enter the details for the new tenant, such as the tenant name, description, the organization name, and the domain name for the tenant, and then click save.

Alternative vManage Deployments

VMWare

Installing vManage on VMWare is very straightforward. In vCenter, click *File* ➤ *Deploy OVF Template…* (Figure 3-35).

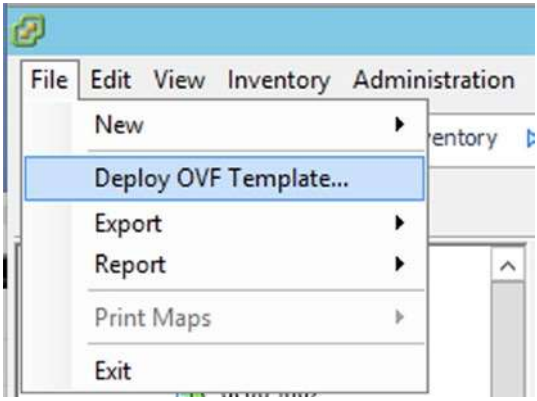

Figure 3-35. *Deploying OVF templates*

Select the viptela-vmanage-19.3.0-genericx86_64.ova file (Figure 3-36).

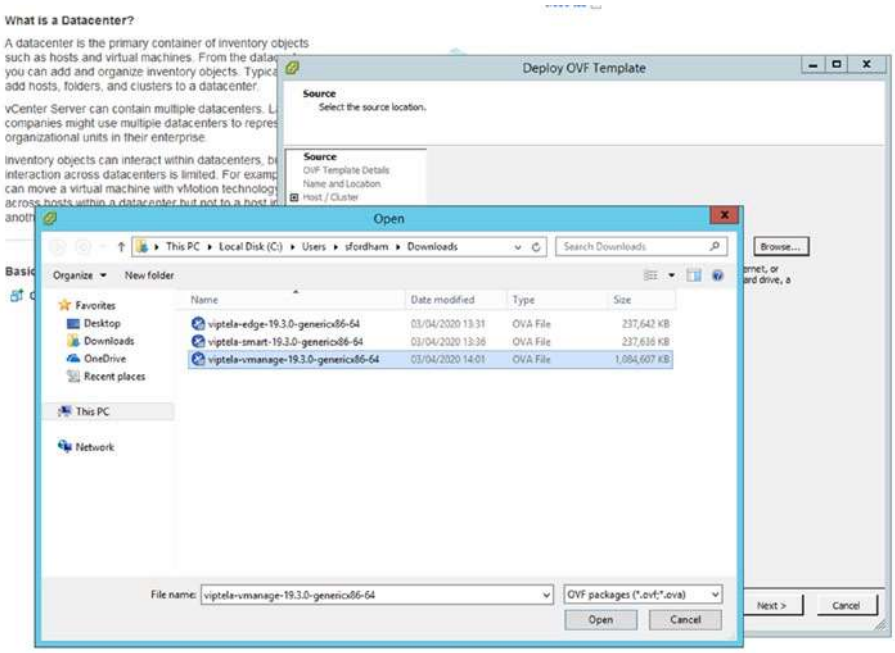

Figure 3-36. *Selecting the OVF*

Click Next to see the OVF details (Figure 3-37).

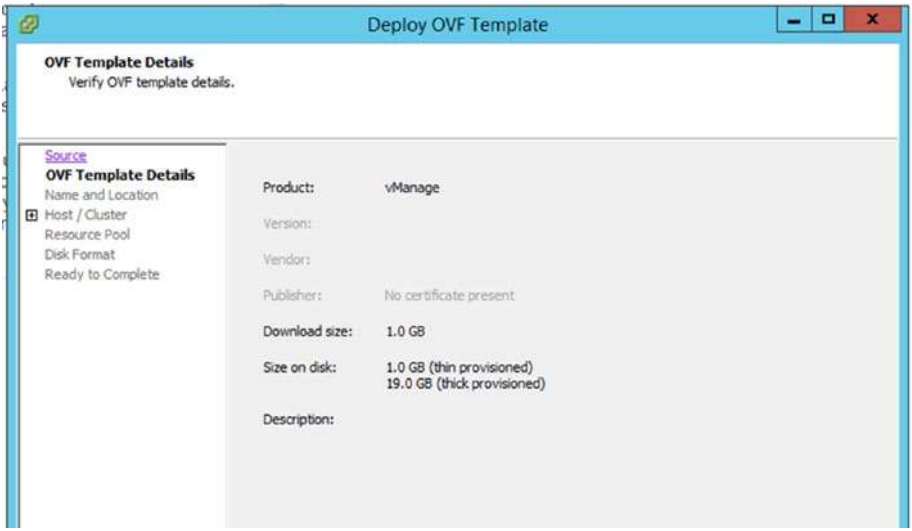

Figure 3-37. *The OVF details*

Name the VM and select a location (Figure 3-38).

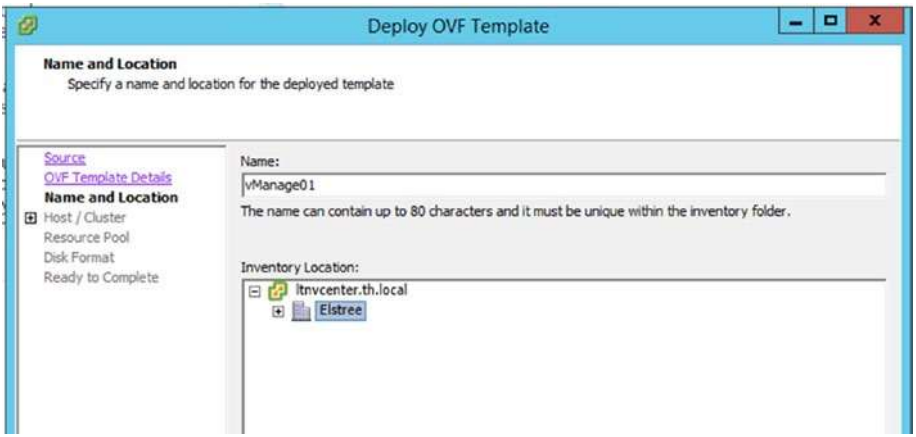

Figure 3-38. *Setting the virtual machine name*

Select an ESXi host (if you have more than one) (Figure 3-39).

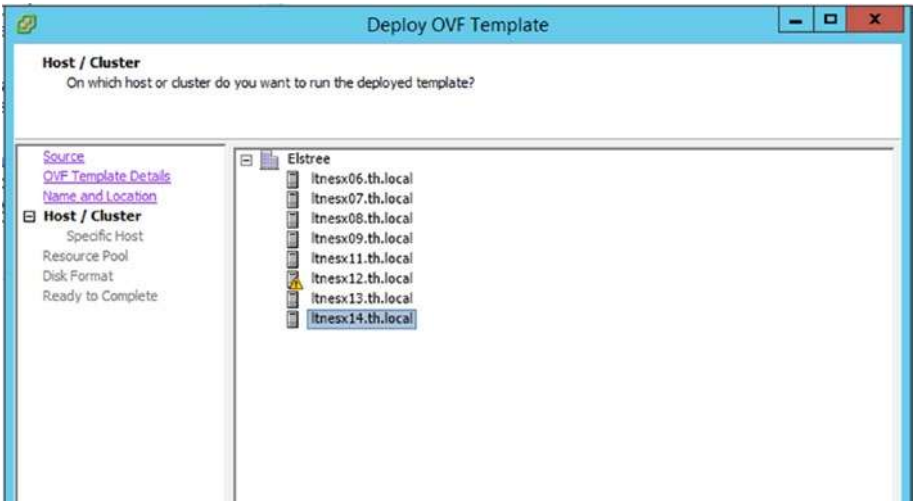

Figure 3-39. *Selecting an ESXi host*

Select a datastore (Figure 3-40).

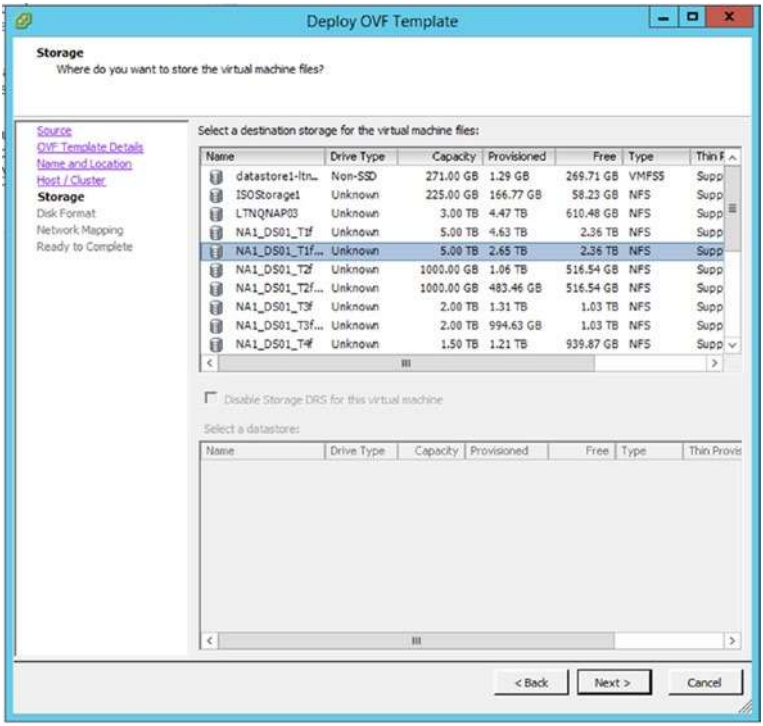

Figure 3-40. *Selecting the datastore*

Click Next through the disk format (Figure 3-41).

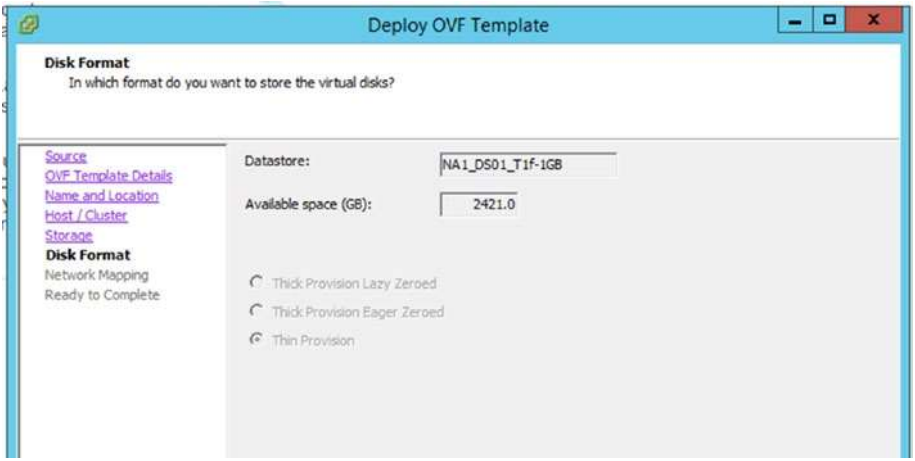

Figure 3-41. *VM disk format*

Select the networks to use (Figure 3-42).

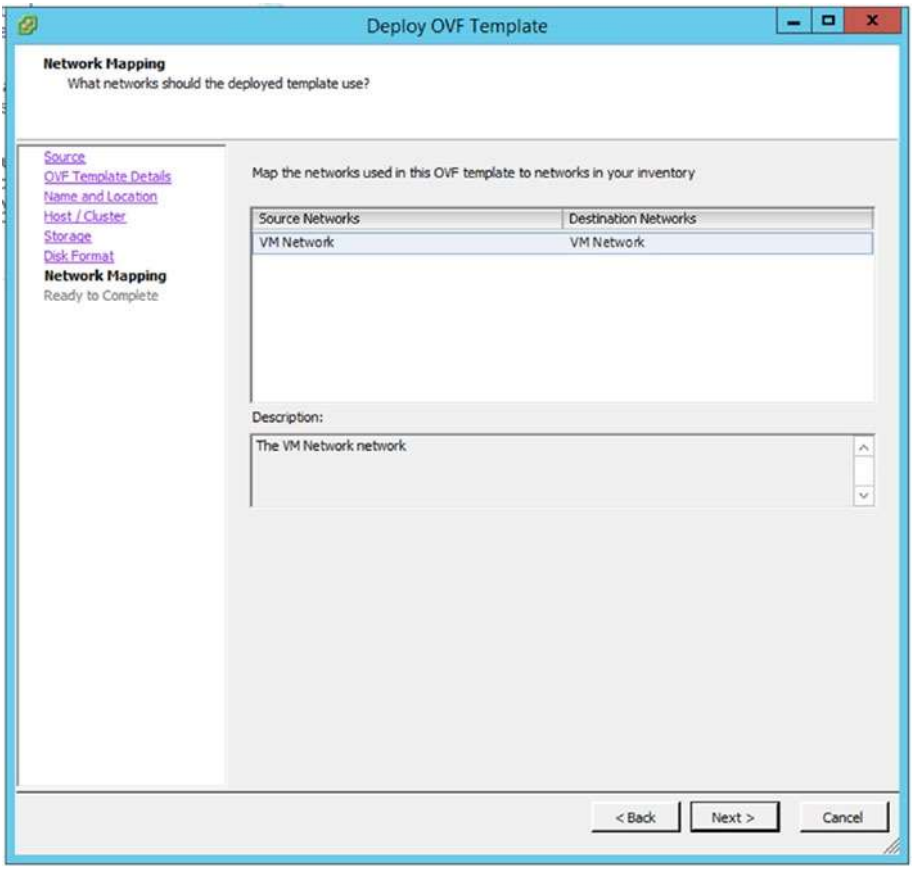

Figure 3-42. *VMWare networks*

Click Finish (Figure 3-43).

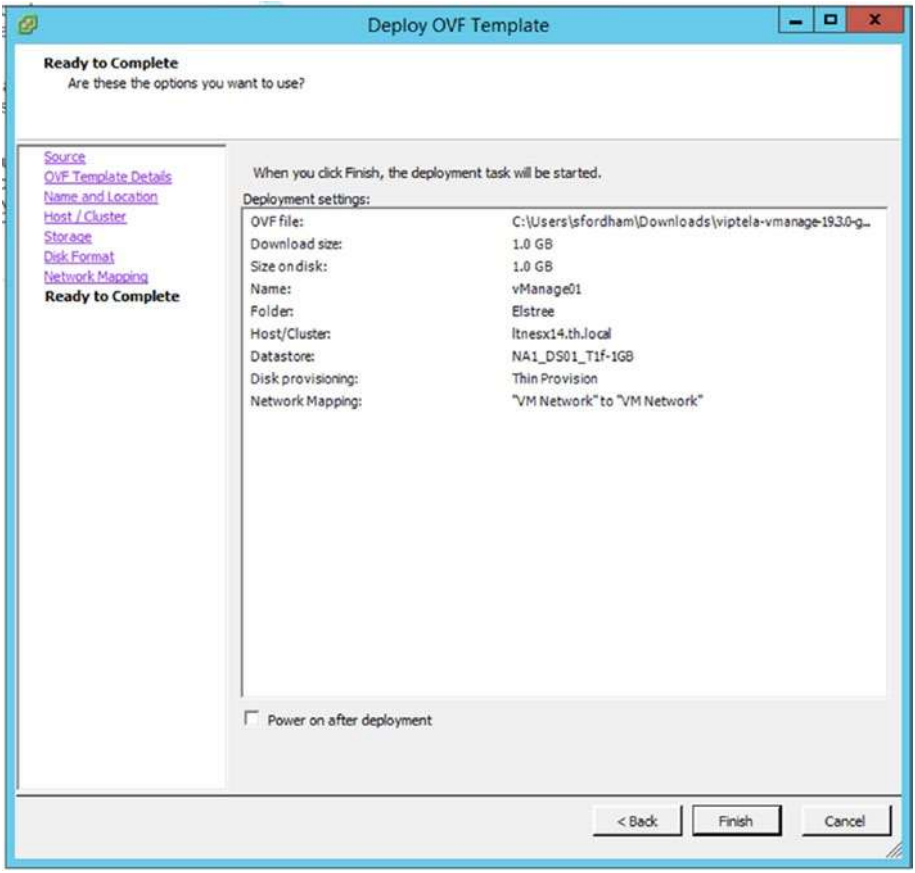

Figure 3-43. *Finishing the setup*

The VM will be deployed (Figure 3-44).

Figure 3-44. *Deploying the vManage server in VMWare*

The next step is to add the second hard disk for storage.

Select the vManage01 VM, and click "Edit virtual machine settings" (Figure 3-45).

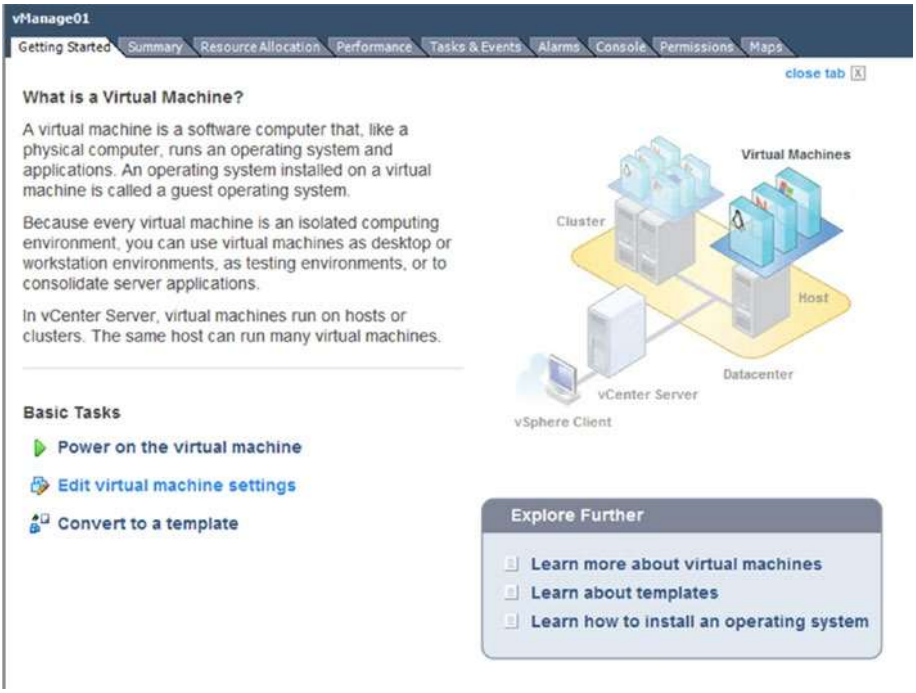

Figure 3-45. *Editing the virtual machine settings*

Click "Add..." (Figure 3-46).

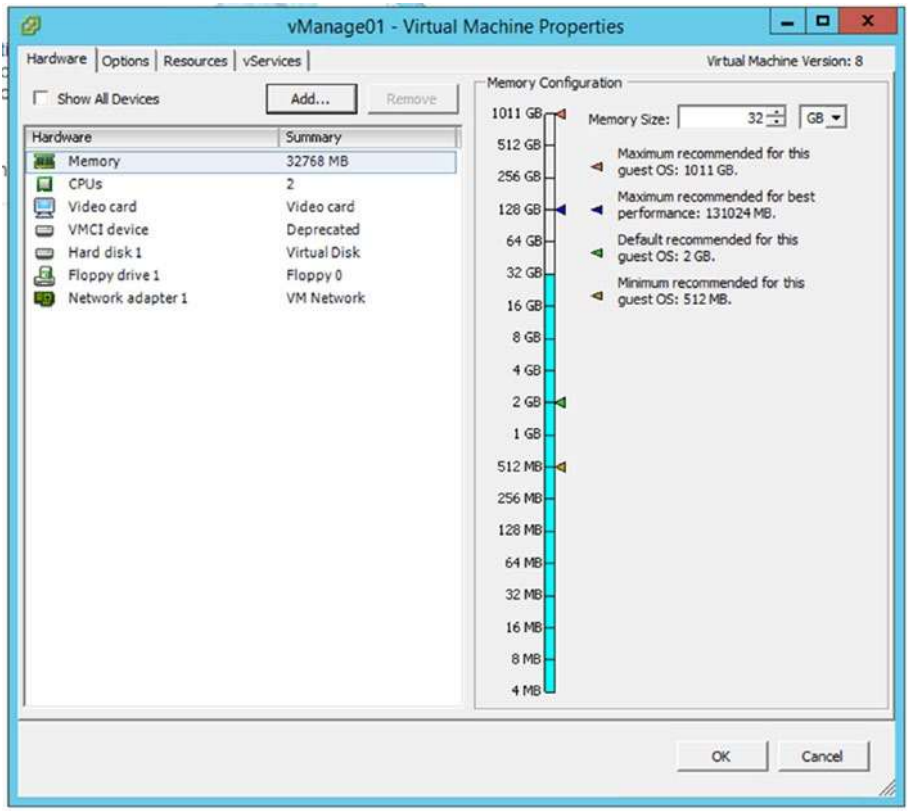

Figure 3-46. *Adding new hardware*

In the Add Hardware dialog box, click Hard Disk (Figure 3-47).

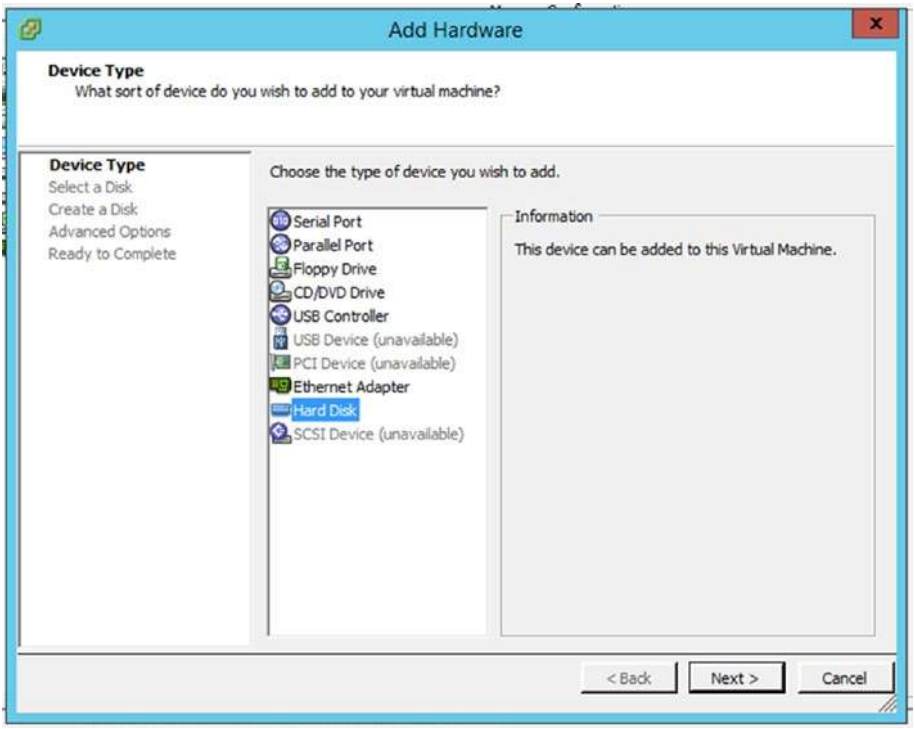

Figure 3-47. *Adding a hard disk*

Click Next to select the option to create a new virtual disk (Figure 3-48).

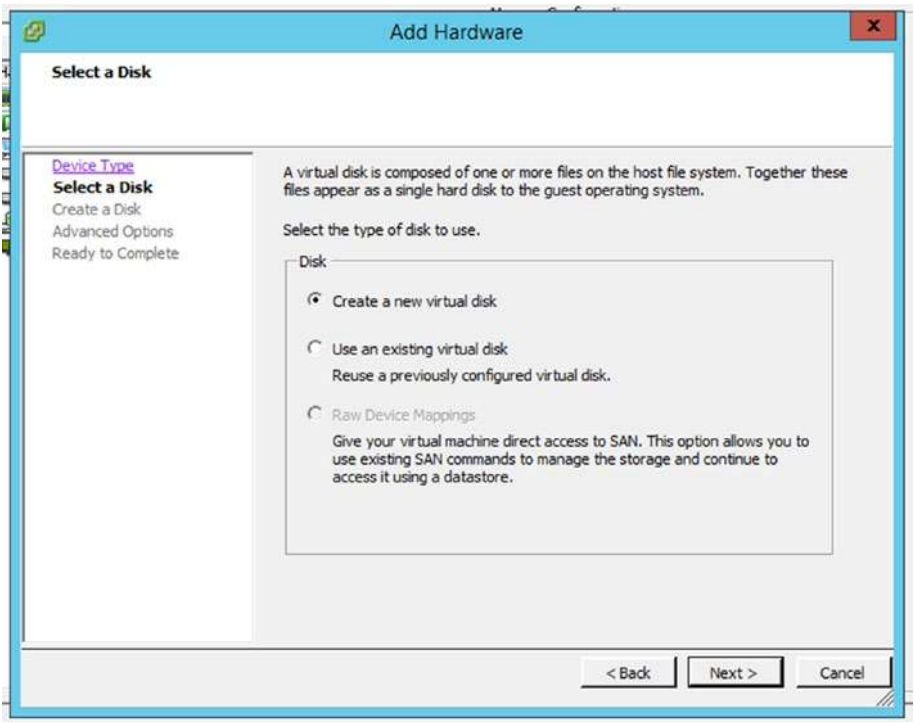

Figure 3-48. *Create a virtual disk*

Set the size to be 100GB, and click Next (Figure 3-49).

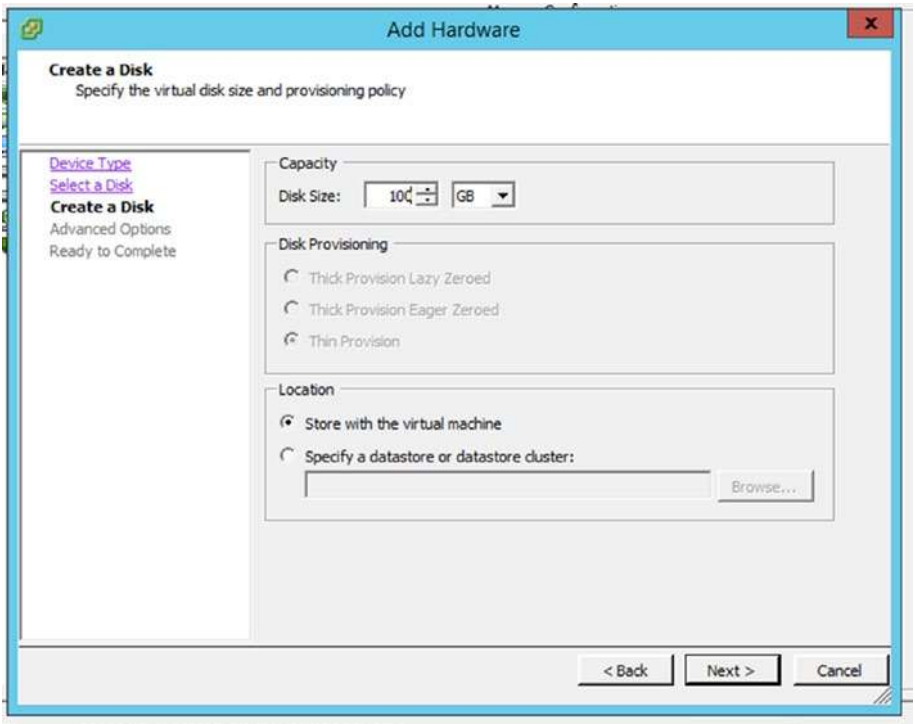

Figure 3-49. *Set the disk size*

Set the Virtual Device Node to be "IDE"; SCSI is not supported (Figure 3-50).

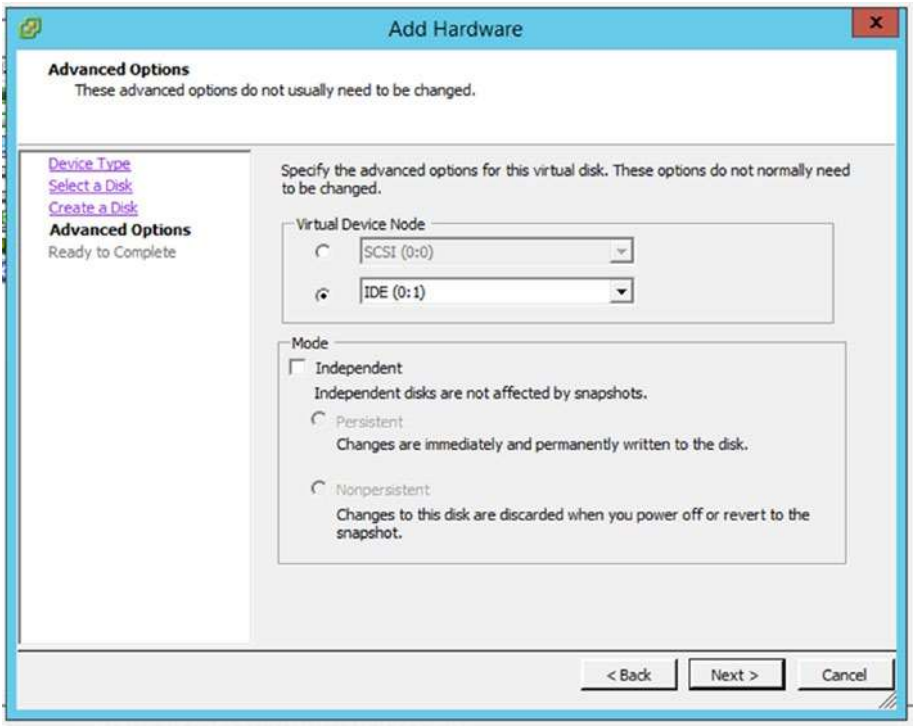

Figure 3-50. *Make sure you select IDE!*

Click Next and then click Finish (Figure 3-51).

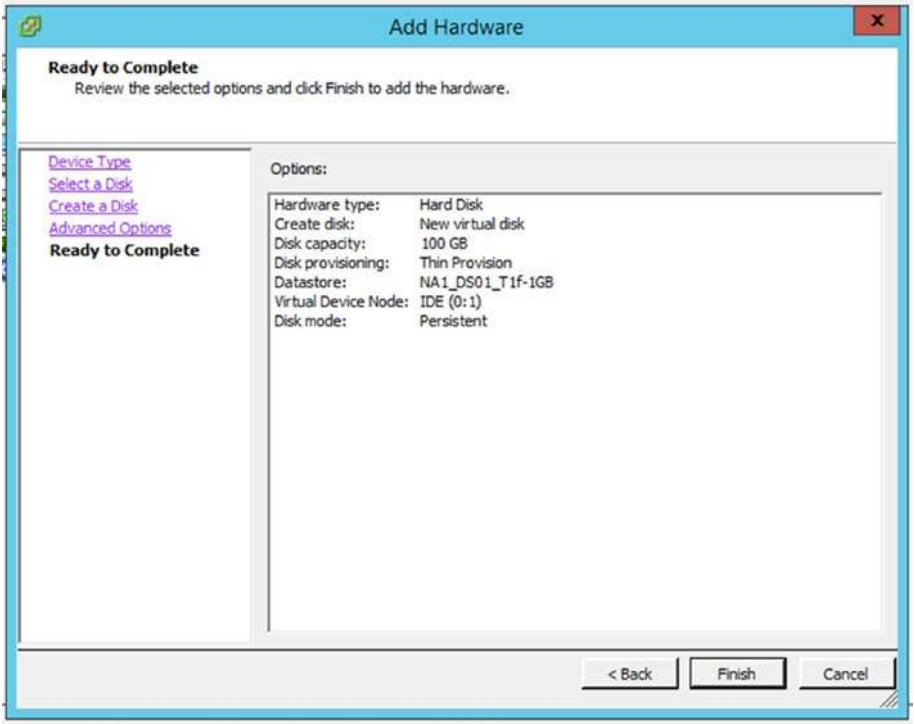

Figure 3-51. *Finishing the additional hard disk*

The next and final stage is to add a second NIC, for the out-of-band management (VPN 512).

Click "Edit virtual machine settings," and then click Add and select "Ethernet Adapter" (Figure 3-52).

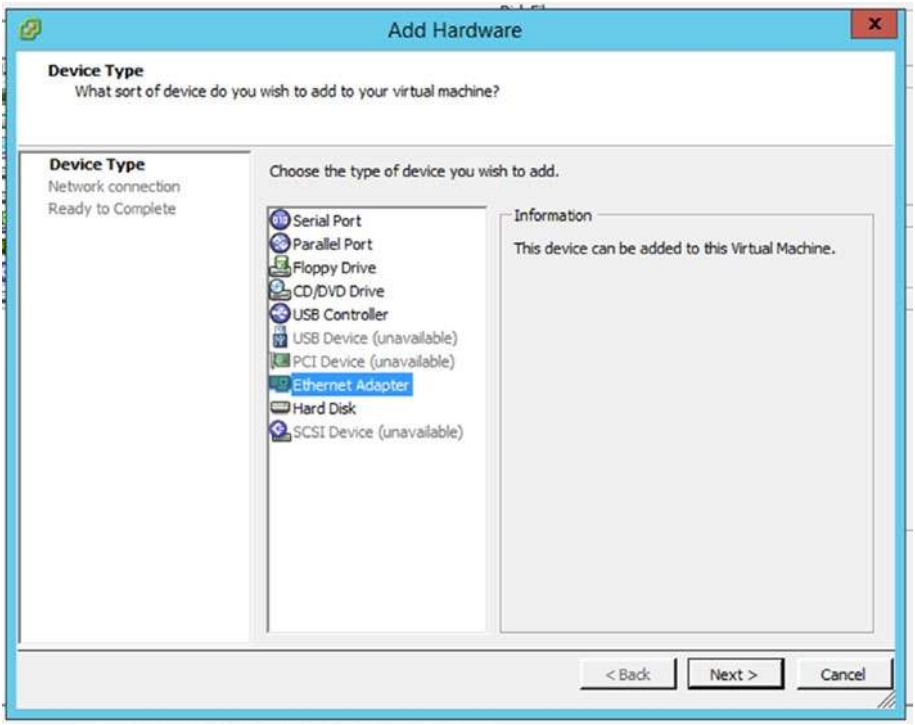

Figure 3-52. *Adding a new Ethernet adapter*

The type should be “VMXNET 3” (Figure 3-53).

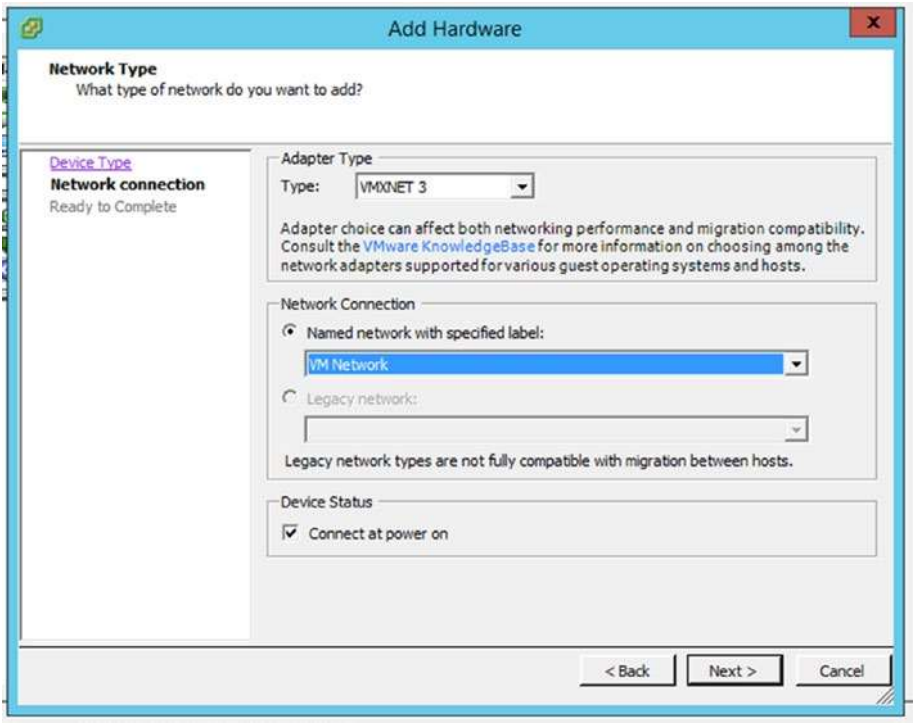

Figure 3-53. *Selecting the network*

Click “Next” and then “Finish” (Figure 3-54).

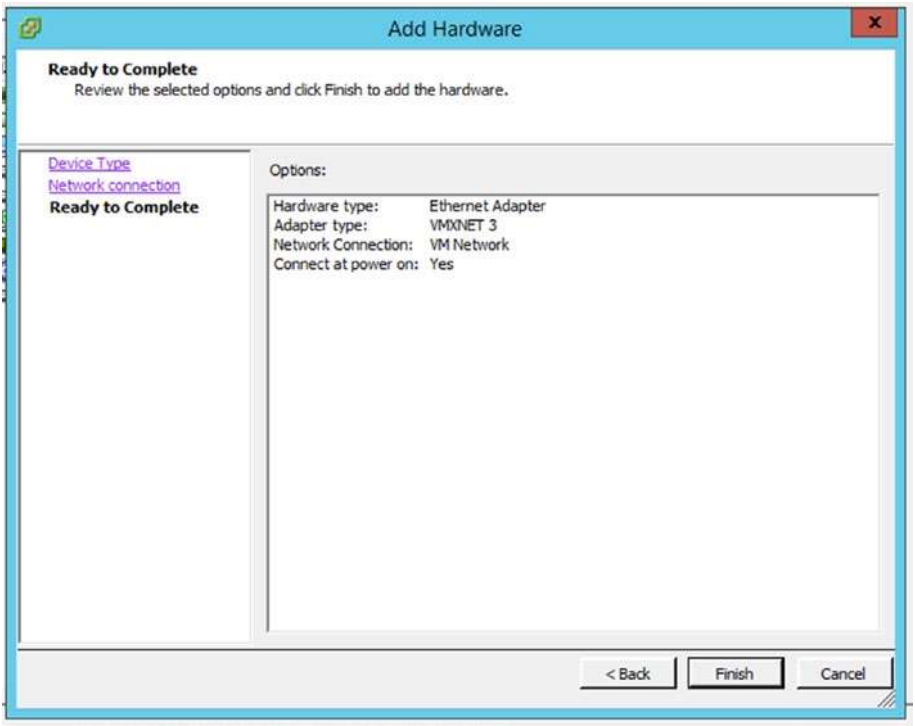

Figure 3-54. *Finishing the vManage hardware changes*

vManage is ready to run on ESXi now.

KVM

Using the qemu-img command, create a second disk:

```
qemu-img create -f qcow2 vmanage-disk2.qcow2 100G
```

Setup vManage as follows:

```
virt-install \
    --name vManage01 \
    --os-type linux \
    --os-variant ubuntu14.04 \
```

```
--cpu host \
--vcpus=2 \
--hvm \
--arch=x86_64 \
--ram 16384 \
--disk path=viptela-vmanage-19.3.0-genericx86-64.qcow2,size=
  16,device=disk,bus=ide,format=qcow2 \
--disk path=vmanage-disk2.qcow2,size=16,device=disk,bus=ide,
  format=qcow2 \
--network=network:default,model=virtio \
--network=network:default,model=virtio \
--graphics none \
--import
```

The Viptela Serial File

In the previous chapter, we downloaded a serial file from the Smart account portal. We now need to get this file into vManage.

The first step is to copy the file over to the VM directory, using something like FileZilla. Once it is there, we should be able to see it along with the virtioa.qcow2 file (I have edited the folder name to make it easier to read the output in the following):

```
root@eve-ng:~# cd /opt/unetlab/addons/qemu/linux-ubun-
dsktp-17.10.1/
root@eve-ng:/opt/unetlab/addons/qemu/linux-ubun-dsktp-17.10.1# ls
serialFile.viptela  virtioa.qcow2
root@eve-ng:/opt/unetlab/addons/qemu/linux-ubun-dsktp-17.10.1#
```

You need to install the MKISOFS utility, which you can do by following the guide on this page: www.802101.com/how-to-get-files-into-qemu-vm/

The next stage is to create a CD ROM for our Linux machine:

```
root@eve-ng:/opt/unetlab/addons/qemu/linux-ubun-dsktp-17.10.1#
mkisofs -o cdrom.iso serialFile.viptela
I: -input-charset not specified, using utf-8 (detected
in locale settings)
Total translation table size: 0
Total rockridge attributes bytes: 0
Total directory bytes: 0
Path table size(bytes): 10
Max brk space used 0
176 extents written (0 MB)
root@eve-ng:/opt/unetlab/addons/qemu/linux-ubun-dsktp-17.10.1# ls
cdrom.iso  serialFile.viptela  virtioa.qcow2
root@eve-ng:/opt/unetlab/addons/qemu/linux-ubun-dsktp-17.10.1#
```

Reboot the Linux VM and the CD Rom should be visible (Figure 3-55).

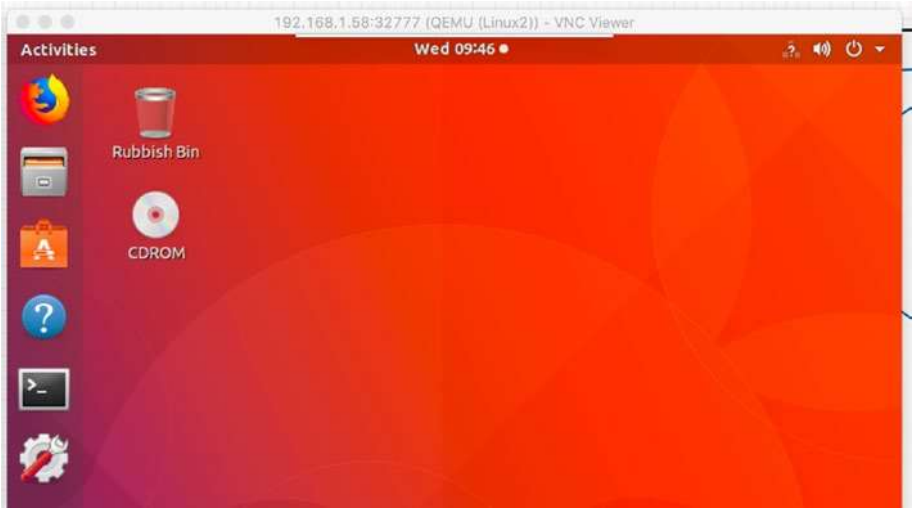

Figure 3-55. *We have a CDROM!*

The serialFile.viptela will have had the name truncated as we used the standard format (ISO 9660) when creating our ISO file, so file names have a maximum of eight characters with a three-character extension. Later on, we'll use Joliet extensions so that this does not happen again. Copy the serialfi.vip file to /tmp, and rename it back to serialFile.viptela (Figure 3-56).

```
user@user-virtual-machine:~$ cp /media/user/CDROM/serialfi.vip /tmp/serialFile.
viptela
user@user-virtual-machine:~$
```

Figure 3-56. *Renaming our serial file*

On vManage01, navigate to *Configuration* ➤ *Devices*, and click "Upload WAN Edge List" (Figure 3-57).

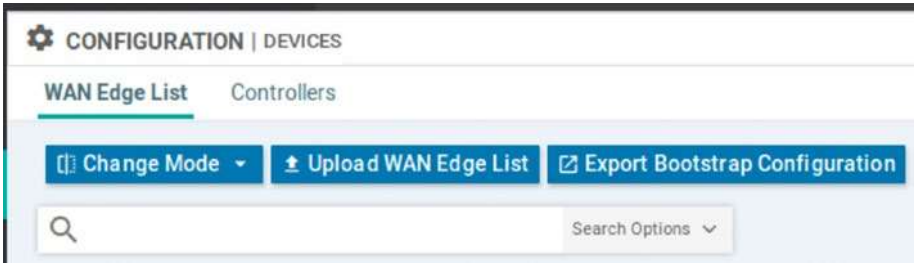

Figure 3-57. *Uploading the WAN edge list*

Select the serialFile.viptela file that is in the /tmp directory (Figure 3-58).

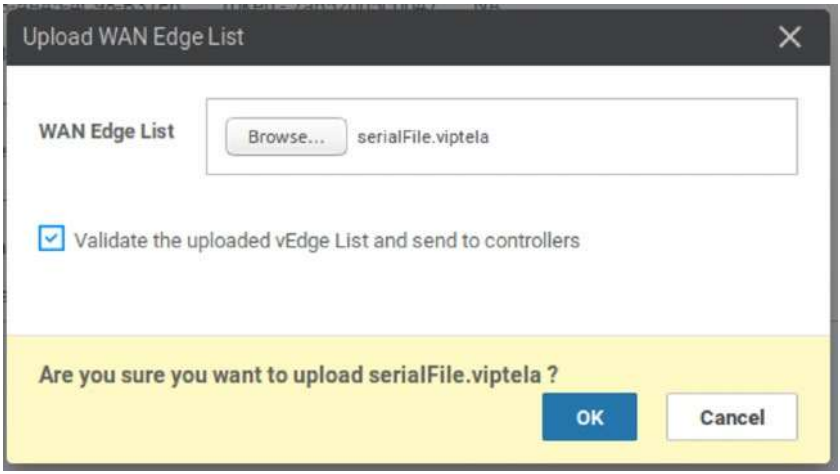

Figure 3-58. *Yes, we are sure*

The file will upload (Figure 3-59).

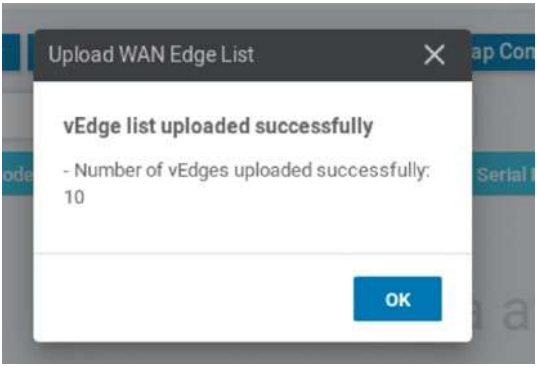

Figure 3-59. *The successful upload*

We can check this in the *Configuration* ➤ *Devices* page (Figure 3-60).

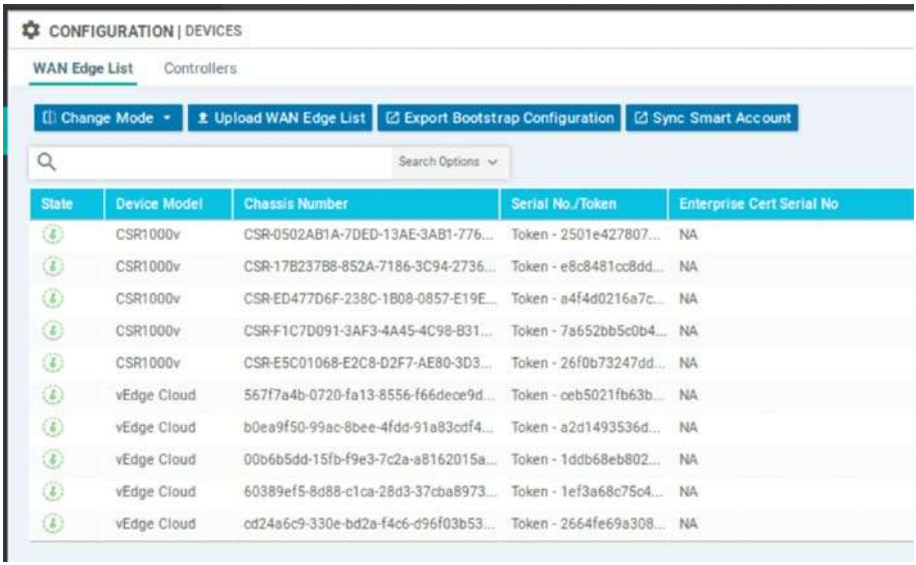

Figure 3-60. *Our edge device list!*

Now that we have our vManage server up and running and loaded with our serial file, we can start to build out the rest of the network.

Summary

In this chapter, we set up our vManage NMS cluster and set the certificate authority using our Linux server in VPN 0. We set up some extra users and looked at tenancy options. We then uploaded the Viptela serial which will enable us to add our edge devices. Next up, we should look at how the SD-WAN network will operate.

CHAPTER 4

Understanding the Overlay

During our setup of the vManage server(s), we created two VPN connections, VPN 0 and VPN 512. These have different purposes; we use VPN 512 for out-of-band management, whereas VPN 0 is our transport VPN, but can also be used for management purposes.

In this chapter, we are going to look at these two VPNs in greater detail and the OMP routing protocol, which forms the overlay network used by the SD-WAN, as well as BFD and NETCONF (Network Configuration Protocol).

VPN 512

This management VPN is enabled by default on all Viptela devices, but will be unconfigured, such as here on the vSmart controller:

```
vSmart01# sh run vpn 512
vpn 512
!
vSmart01#
```

Typically, on vEdge devices, the Gigabit Ethernet interface will be used; on other devices, such as vBond, the Ethernet interface will be used.

The IP address configuration can be configured for static or DHCP IP addressing:

```
vBond01# sh run vpn 512
vpn 512
 interface eth0
  ip address 100.1.1.3/24
  ipv6 dhcp-client
  no shutdown
 !
!
vBond01#
```

Apart from a little bit of routing, there is little we can do with VPN 512, so let's look at VPN 0 instead.

VPN 0

VPN 0 is where all the magic happens; this is the WAN transport VPN. All the control plane traffic is carried by this VPN through the overlay network, within OMP sessions.

Note If a device is to be part of the overlay network, then at least one interface must be connected to VPN 0.

We start by defining the interface to be used for VPN 0 and then set the IP address; this address can be IPv4 or IPv6 (or both if you are so inclined):

```
vBond01# sh run vpn 0
vpn 0
 interface ge0/0
  ip address 10.1.1.3/24
  ipv6 dhcp-client
```

Next, we define our tunnel interface. On the vBond and vEdge devices, we need to set an encapsulation method, which can be IPSec or GRE (this is for the TLOC, or Transport Location). We do not need to specify the encapsulation on all devices. It is mandatory on the vBond device, but not required (or even available as a command) on the vSmart or vManage devices.

```
tunnel-interface
 encapsulation ipsec
```

Within the tunnel interface, we specify the allowed services. Routing-wise we can enable BGP and OSPF. To overcome issues with NAT, we can enable the STUN (Session Traversal Utilities for NAT) protocol. For management, we can enable SSH (sshd), NTP, NETCONF, ICMP, HTTP, DNS, and DHCP.

We can enable all of the services (using the command "`allow-service all`"), but even so, will still see those commands that have not been explicitly permitted or denied showing in the configuration as having their default values. Hopefully, this will be changed in future versions:

```
 allow-service all
 no allow-service bgp
 allow-service dhcp
 allow-service dns
 allow-service icmp
 no allow-service sshd
 no allow-service netconf
 no allow-service ntp
 no allow-service ospf
 no allow-service stun
 allow-service https
!
```

Access such as SSH will be available over VPN 512 by default; however, we can also enable this over VPN 0. NETCONF should be enabled on edge devices as this is how vManage provisions virtual devices, as well as the edge devices using NETCONF to send notifications back to vManage.

The VPN 0 interface will, by default, be placed in a shutdown state. We need to enable it.

```
  no shutdown
 !
```

Lastly, we (may) need to provide routing information, which is done under the interface, where we set the IP address:

```
 ip route 0.0.0.0/0 10.1.1.1
!
vBond01#
```

The options available under the tunnel interface differ depending on the platform. There are, however, some constants, such as the allowed services, the Hello interval, and Hello tolerance, which set the time (in seconds) that we send hello packets and the control tolerance of these packets.

The Hello interval is the time between Hello packets. These packets are used to check that tunnels between devices are still active and act to keep the tunnel alive. These are sent every one second, by default. If no Hello packet is received, then we use the Hello tolerance to keep the tunnel up that little bit longer. The Hello tolerance is set to 11 seconds (again by default), so if no Hello packet is received within 12 seconds (the original Hello interval plus the tolerance), the tunnel is taken down.

If the interval and tolerance have different values at each end of the DTLS tunnel, then the *lower* hello interval and *higher* tolerance interval will be used between controller devices (vManage, vBond, and vSmart). If one side of the tunnel is an edge router, then the tunnel will use the values configured on the router. This is to minimize the amount of traffic sent over the tunnel.

DTLS

The Viptela/Cisco SD-WAN uses DTLS or TLS tunnels between devices (Figure 4-1). DTLS (Datagram Transport Layer Security) is based on TLS and was designed to do what SSL could not, namely, create a secure protocol under UDP. The vBond will *only* use DTLS, though.

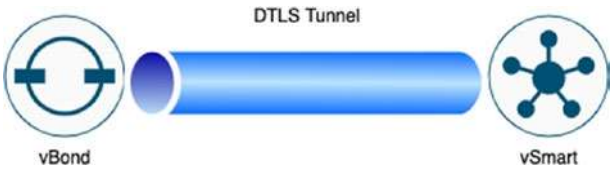

Figure 4-1. *A DTLS tunnel*

TCP has measures built into it to make it reliable. It will request missing packets, and it will reorder packets received so they are in the correct order. These are all excellent ways of giving us a network we can use. However, once we start putting TCP within TCP (like in tunnels), then these reliability measures can be detrimental. This is referred to as "*TCP meltdown*."

With TCP meltdown, when the underlying protocol has an issue, it tries to recover (by requesting a missing packet, for example); this can cause the preceding layer to also compensate, which can, in turn, cause delays on the network.

To avoid this, UDP is used instead. While UDP is connectionless and lacks recovery mechanisms, issues with loss of datagrams and out-of-order packets have to be offloaded to the application (SD-WAN), instead of having TCP do this for us. But at least we avoid TCP meltdown.

Within the tunnel are the OMP messages (Figure 4-2).

OMP

OMP is the Overlay Management Protocol. This is the control plane of the SD-WAN and runs between the vEdges and the vSmart controllers.

Figure 4-2. *DTLS tunnel carrying OMP messages*

OMP performs the following functions:

- Network overlay orchestration, such as site connectivity
- Distribution of routing information and location mappings (the TLOCs)
- Data-plane security parameter distribution
- Control and distribution of routing policy

OMP is enabled by default, as we can see on the vSmart controller:

```
vSmart01# show omp summary
oper-state              UP
admin-state             UP
personality             vsmart
omp-uptime              4:19:03:49
routes-received         0
routes-installed        0
routes-sent             0
tlocs-received          0
tlocs-installed         0
```

```
tlocs-sent              0
services-received       0
services-installed      0
services-sent           0
mcast-routes-received   0
mcast-routes-installed  0
mcast-routes-sent       0
hello-sent              0
hello-received          0
handshake-sent          0
handshake-received      0
alert-sent              0
alert-received          0
inform-sent             0
inform-received         0
update-sent             0
update-received         0
policy-sent             0
policy-received         0
total-packets-sent      0
total-packets-received  0
vsmart-peers            0
vedge-peers             0
vSmart01#
```

OMP advertises the following:

- OMP routes/vRoutes
- Service routes
- TLOCs

To understand these three different types of routes a little easier, we can break our network into two halves: the service side and the transport side.

OMP Routes/vRoutes

OMP routes are also known as vRoutes; these will be the service prefixes mentioned earlier. These routes are advertised to the vSmart controller as they are collected from the edge devices. Any connected, static, OSPF inter-area or intra-area routes will be automatically injected into OMP. BGP and OSPF external routes will need to be manually redistributed within OMP.

Looking at our topology, in Figure 4-3, the lines in black would be our OMP routes.

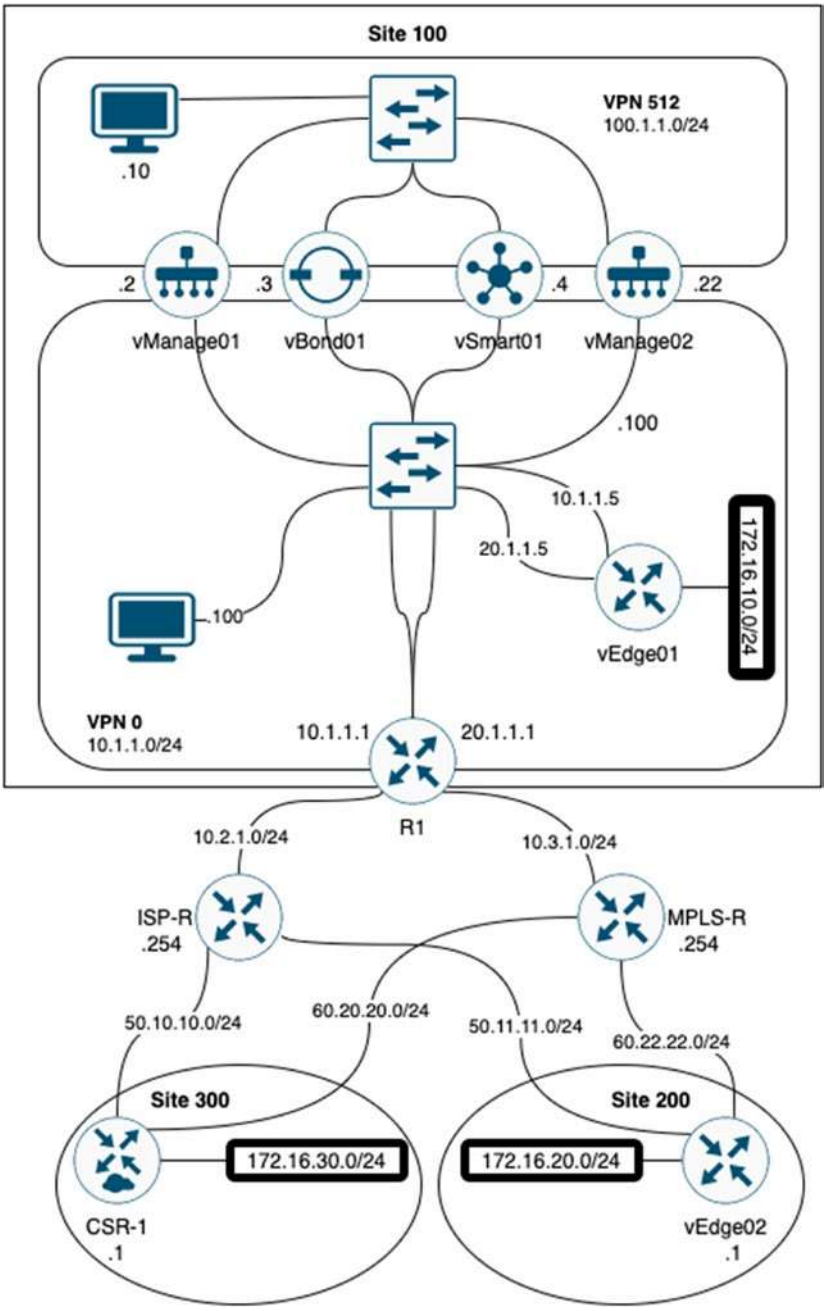

Figure 4-3. *Our OMP routes*

Service Routes

Service routes are identifiers that link an OMP route to a service on a vEdge router or within the site that the vEdge router is in. These services could be a firewall, IPS/IDS, or a load balancer, for example. These routes specify the location of the service. We do not have any of these in our topology, but if we did (such as a firewall inside site 100), the service routes would look like in Figure 4-4.

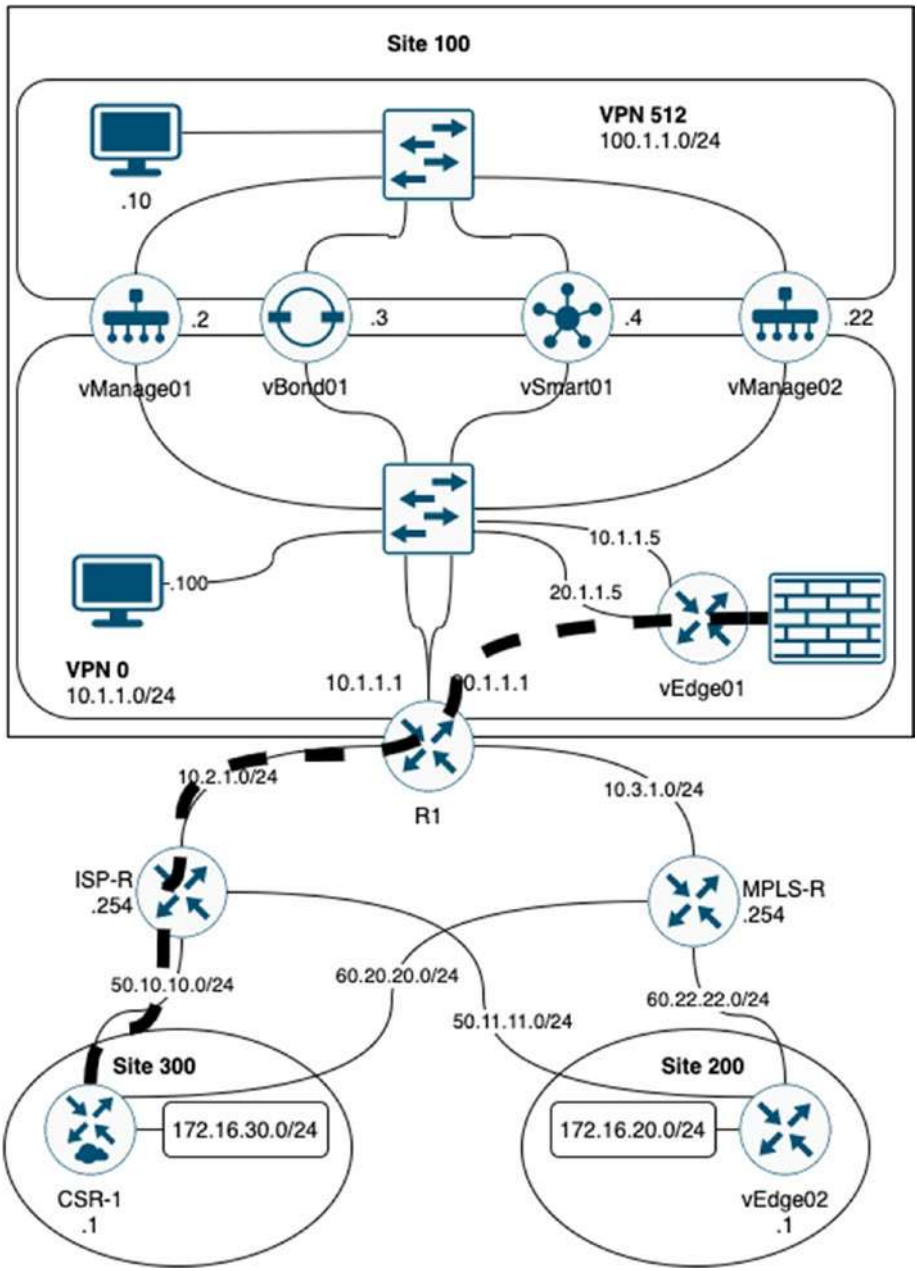

Figure 4-4. *Service routes*

Service routes use the Subsequent Address Family Identifier (SAFI), which include the following attributes:

- VPN ID
- Service ID
- FW
- IDS
- IDP
- Generic net-svc
- Label
- Originator ID
- TLOC
- Path ID

TLOC

TLOC stands for Transport Location, and it is a way of separating the route from the endpoint on which it sits; routes are based on location, not on specific router interfaces. TLOCs are our next hops and WAN attachment points, they are the only part of the OMP that is visible to the physical network, and these TLOCs must be reachable by the routing domain.

Figure 4-5 shows these TLOCs.

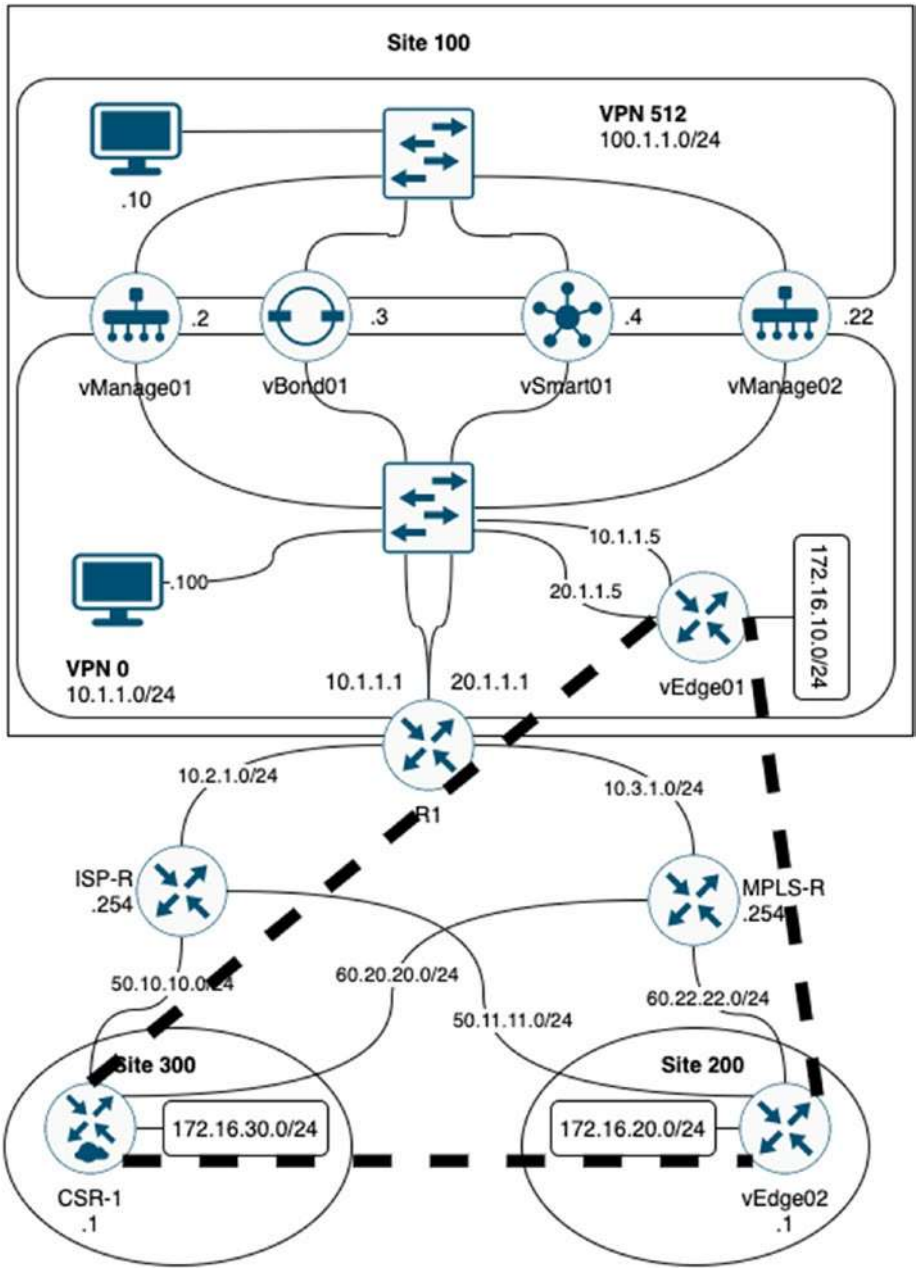

Figure 4-5. *TLOCs*

A TLOC has three components:

- System IP. We have seen these already when we set up our vManage NMS. This "IP" does not have to be routable or reachable; it is for identification only, similar to a BGP router ID. It is not an IP in the proper sense, it is purely a dotted decimal set of numbers. We set this in the system properties.
- Color. The color is used to distinguish between different transports.
- Encapsulation type, which will either be GRE or IPSec.

The TLOC will advertise the following:

- TLOC private address
- TLOC public address
- Carrier
- Color
- Encapsulation type
- Preference
- Site ID
- Tag
- Weight

The carrier attribute is an identifier, and there are eight options to choose from (carrier1 through to carrier8). The carrier attribute is used, when we are using NAT and private colors, to control whether we use the

private or public IP address for session establishment. If the carrier setting is the same, then the private IP address is used. If the carrier setting is different, then then public IP address is used:

```
vBond01(config-tunnel-interface)# carrier ?
Description: Set carrier for TLOC
Possible completions:
  <default carrier1 carrier2 carrier3 carrier4 carrier5
carrier6 carrier7 carrier8>[default]
vBond01(config-tunnel-interface)#
```

The color identifies the link type from a predefined list:

```
vBond01(config-tunnel-interface)# color ?
Description: Set color for TLOC
Possible completions:
  <3g  biz-internet  blue  bronze  custom1  custom2  custom3
  default  gold  green  lte  metro-ethernet  mpls public-
  internet  red  silver private1 private2 private3 private4
  private5 private6>[default]
vBond01(config-tunnel-interface)# color
```

Color does more than link identification; it defines the VPN tunnel establishment logic, as we will see later. Color is an important area to understand, so we will go into this in greater detail in Chapter 9 when we can see it in action.

Preference is used to differentiate OMP routes advertised by two different TLOCs; the TLOC with the highest preference will be advertised out. If an OMP route is reachable through two (or more TLOCs), then we can use the weight attribute to control our outbound traffic.

Note By default, traffic will be balanced equally across multiple TLOCs, and each of these will have a value of 1. If this value is increased (to a maximum of 255), then more traffic will be sent to this TLOC. We can use this to perform unequal cost multi-pathing, for example, to send more flows across higher-bandwidth lines, by configuring one TLOC with a weight of 100 and the other with a weight of 10, to get a 10:1 traffic ratio.

The tag is used for TLOC filtering; this is an optional, transitive path attribute.

We will look closer at TLOCs in Chapter 7 when we set up our edge routers.

BFD

BFD, or Bidirectional Forwarding Detection, is used to quickly detect path failures. BFD is used by several technologies such as OSPF and BGP as well as SD-WAN.

BFD sessions are created automatically when edge routers come up and they run inside IPSec connections:

```
vEdge01# show bfd summary
sessions-total         1
sessions-up            1
sessions-max           2
sessions-flap          2
poll-interval          600000
vEdge01# show bfd tloc-summary-list
```

```
IF              SESSIONS  SESSIONS  SESSIONS
NAME    ENCAP   TOTAL     UP        FLAP
-----------------------------------------
ge0/0   ipsec   1         1         2

vEdge01#
```

We can use the "`show bfd history`" and "`show bfd sessions`" commands to look at the BFD details. The history command (Figure 4-6) shows us the state changes and when they occurred.

```
vEdge01# show bfd history
                                                DST PUBLIC      DST PUBLIC                                  RX      TX
SYSTEM IP        SITE ID  COLOR          STATE  IP              PORT        ENCAP  TIME                     PKTS    PKTS    DEL
-------------------------------------------------------------------------------------------------------------------------------
60.100.100.1     200      red            down   50.11.11.1      12426       ipsec  2020-04-09T13:44:15+0000 4       5       0
60.100.100.1     200      red            up     50.11.11.1      12426       ipsec  2020-04-09T13:44:16+0000 1014693 1014717 0
60.100.100.1     200      red            down   50.11.11.1      12426       ipsec  2020-04-15T10:36:32+0000 1014698 1014765 0
60.100.100.1     200      red            up     50.11.11.1      12426       ipsec  2020-04-15T10:37:16+0000 1014698 1014765 0
60.100.100.1     200      red            down   50.11.11.1      12426       ipsec  2020-04-15T10:37:16+0000 0       0       0
60.100.100.1     200      red            down   50.11.11.1      12426       ipsec  2020-04-15T10:37:16+0000 0       0       0
60.100.100.1     200      red            down   50.11.11.1      12346       ipsec  2020-04-15T10:37:19+0000 10      10      0
60.100.100.1     200      red            up     50.11.11.1      12346       ipsec  2020-04-15T10:37:20+0000 0       0       0

vEdge01#
```

Figure 4-6. *BFD history*

The sessions command (Figure 4-7) shows us our current sessions.

```
vEdge01# show bfd sessions
                                                SOURCE TLOC     REMOTE TLOC                    DST PUBLIC      DST PUBLIC                  DETECT      TX
SYSTEM IP        SITE ID  STATE                 COLOR           COLOR           SOURCE IP      IP              PORT        ENCAP           MULTIPLIER  INTERVAL(msec)  UPTIME          TRANSITIONS
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
60.100.100.1     200      up                    blue            red             10.1.1.5       50.11.11.1      12346       ipsec           7           1000            2:01:35:10      0

vEdge01#
```

Figure 4-7. *BFD sessions*

BFD is used to confirm that remote TLOCs are active. We have one BFD session per TLOC *per destination TLOC,* and if the BFD session fails, then the vSmart controller will remove all the routes which point to that particular TLOC as a next hop.

NETCONF

NETCONF (Network Configuration Protocol) is the last topic for this chapter. NETCONF is a "*simple mechanism through which a network device can be managed*" (RFC 4741[1]). It allows us to pull configuration data out and push configuration data in and uses XML to encode a remote procedure call (RPC).

The RFC for NETCONF (RFC 4741) was published in 2006 by Rob Enns, then of Juniper Networks. We then have another couple of useful RFCs, such as RFC 4742[2] for using NETCONF over SSH and RFC 5539[3] for NETCONF over TLS.

NETCONF can be seen as four different layers:

- Content (configuration data and notification data)
- Operations (retrieve and edit the configuration data)
- Messages (for encoding RPCs and notifications)
- Secure transport (secure and reliable transport of messages between a client and a server)

The basic NETCONF operations are shown in Table 4-1.

[1] https://tools.ietf.org/html/rfc4741
[2] https://tools.ietf.org/html/rfc4742
[3] https://tools.ietf.org/html/rfc5539

Table 4-1. *NETCONF commands*

Operation	Purpose
<get>	Retrieve the running configuration and device state information
<get-config>	Retrieve all or part of a specific configuration datastore
<edit-config>	Edit the configuration through creating, deleting, merging, or replacing
<copy-config>	Copy the configuration datastore to another configuration datastore
<delete-config>	Delete a configuration datastore
<lock>	Lock the configuration datastore of a device
<unlock>	Unlock a locked configuration datastore
<close-session>	Close a NETCONF session (gracefully)
<kill-session>	Close a NETCONF session (ungracefully)

When we send configurations to our edge devices, these configurations are sent as NETCONF messages, within IPSec tunnels. NETCONF is also used to send the signed certificate to the vEdge devices during the onboarding process and then to push localized policies from vManage to the vEdge devices.

Summary

We have looked at the overlay network, including VPN 512 and 0, and covered the DTLS tunnels and the OMP messages carried within them. We then looked at BFD and NETCONF. In the next chapter, we will set up our vBond controller.

CHAPTER 5

Deploying vBond

In this chapter, we are going to set up the vBond server. The basic setup is very similar to the vManage, as all the devices share the same basic configuration.

As with all the devices, you will be prompted to change the password for the admin account when you first log in.

Because vBond and vEdge share the same software image, the device will default to the hostname of "vedge" when it starts up for the first time.

Basic vBond Configuration

Start by entering the hostname, system IP, site ID, and organization name:

```
vedge# config
Entering configuration mode terminal
vedge(config)# system
vedge(config-system)# host-name vBond01
vedge(config-system)# system-ip 100.100.1.3
vedge(config-system)# site-id 100
vedge(config-system)# organization-name "Learning_SD-WAN"
vedge(config-system)#
```

Next, set the vBond IP address. This must match the IP address we set on our VPN 0 interface. We also specify that we are the "`local`" vBond controller. Commit your configuration.

```
vedge(config-system)#
vedge(config-system)# vbond 10.1.1.3 local
vedge(config-system)# commit
```

Note You may see some documentation also using the option of "`vbond-only`" as well as "`local`." You can still use this option, and it will work in the configuration, but it was deprecated in Viptela version 16.2.

In a real-life deployment, the vBond IP address should be a publicly reachable IP address. Next, we move on to our network configuration.

vBond Network Configuration

Type the following:

```
vBond01(config-system)# vpn 0
vBond01(config-vpn-0)# ip route 0.0.0.0/0 10.1.1.1
vBond01(config-vpn-0)# interface ge0/0
vBond01(config-interface-ge0/0)# ip address 10.1.1.3/24
vBond01(config-interface-ge0/0)# no tunnel-interface
vBond01(config-interface-ge0/0)# commit and-quit Commit complete.
vBond01#
```

We need to remove the tunnel interface from VPN 0; otherwise, we will not be able to add vBond to vManage, and we will just get a Java exception error.

We should also set up our management connectivity:

```
vBond01# conf t
Entering configuration mode terminal
vBond01(config)# vpn 512
```

```
vBond01(config-vpn-512)# interface eth0
vBond01(config-interface-eth0)# ip address 100.1.1.3/24
vBond01(config-interface-eth0)# no shutdown
vBond01(config-interface-eth0)# exit
vBond01(config-vpn-512)# commit and-quit
Commit complete.
vBond01#
```

We should now be able to copy over our CA certificate, so, from the Linux server's command prompt, CD into the Downloads directory and type, filling in the authentication details when prompted:

```
scp CA.crt admin@10.1.1.3:
```

Switch back to the vBond01 VM and install the certificate:

```
vBond01# request root-cert-chain install /home/admin/CA.crt
Uploading root-ca-cert-chain via VPN 0
Copying ... /home/admin/CA.crt via VPN 0
Updating the root certificate chain..
Successfully installed the root certificate chain
vBond01#
```

We should now be able to add the vBond device to vManage.

Adding vBond to vManage

Firstly, make sure that vManage has been set up with a vBond device.

Go to *Administration* ➤ *Settings*, and edit the vBond configuration, adding the IP address of our vBond controller. This must be the VPN 0 address we configured earlier (Figure 5-1).

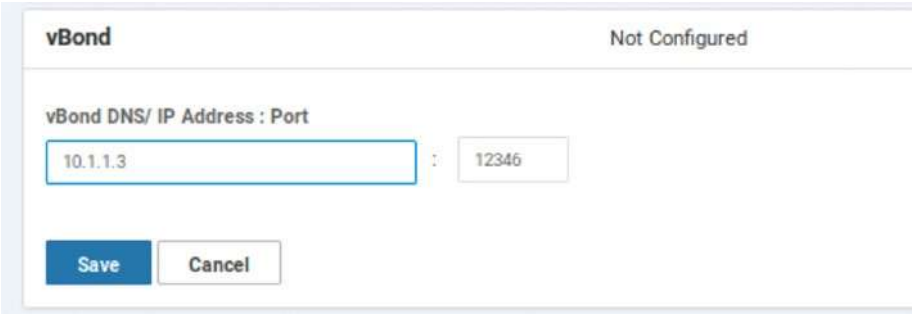

Figure 5-1. *Adding the IP address of the vBond controller to vManage*

Click Save.

Next, navigate to *Configuration* ➤ *Devices* (Figure 5-2).

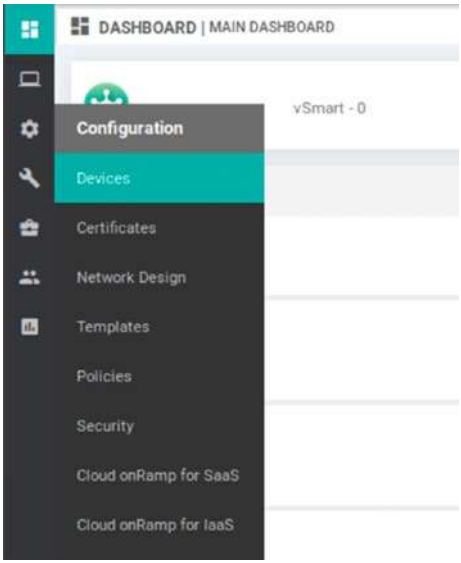

Figure 5-2. *The Devices menu*

Click "*Controllers*" (Figure 5-3).

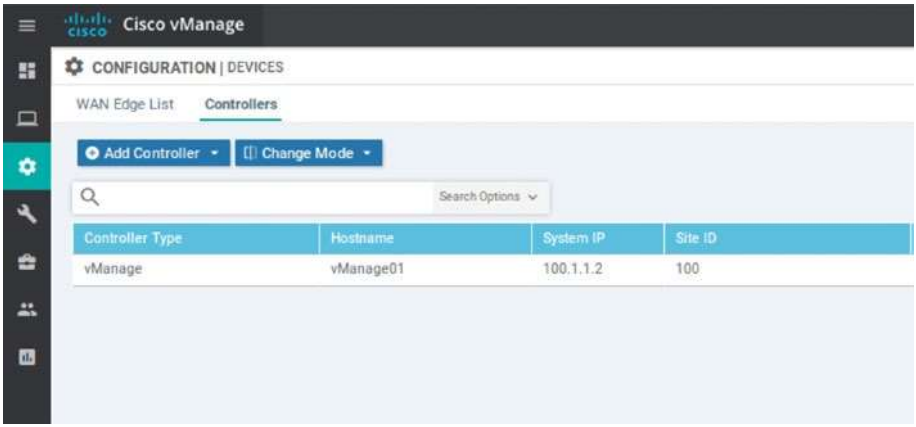

Figure 5-3. *Controllers*

Click "*Add Controller*," and select the vBond option (Figure 5-4).

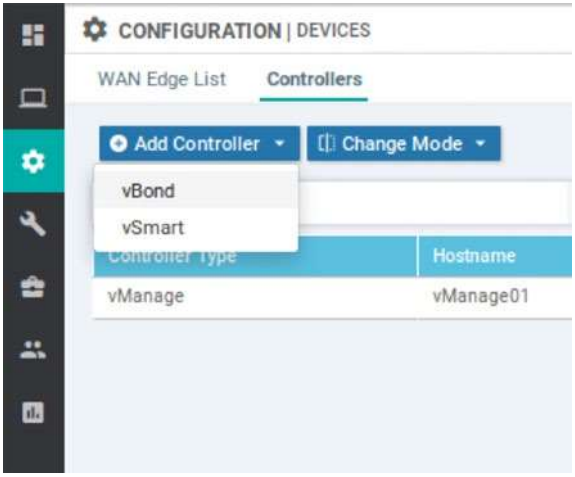

Figure 5-4. *Adding a new controller*

Enter the IP address and username and password of the vBond device, and click "*Add*" (Figure 5-5).

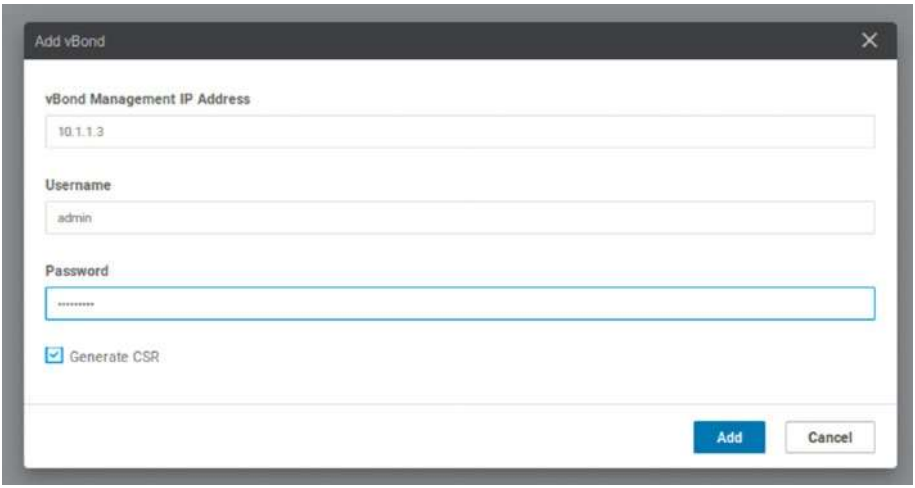

Figure 5-5. *The name is Bond. vBond*

We should see the vBond controller added to our list now as seen in Figure 5-6.

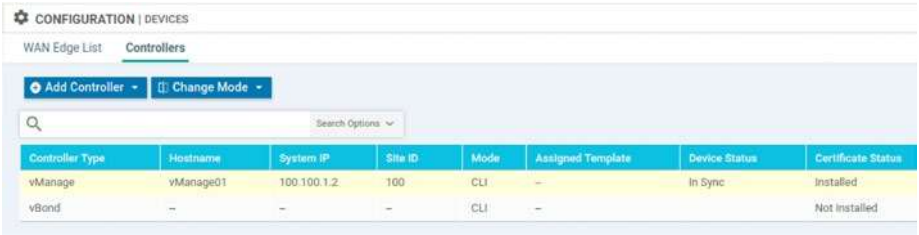

Figure 5-6. *Our controller list*

We still have some work to do, as the vBond server details are missing, and we do not have a certificate installed.

Navigate to *Configuration* ➤ *Certificates* ➤ *Controllers*. The vBond controller will have an operation status of "CSR generated," so click the triple dots and select "View CSR." Download it to the Linux server (Figure 5-7).

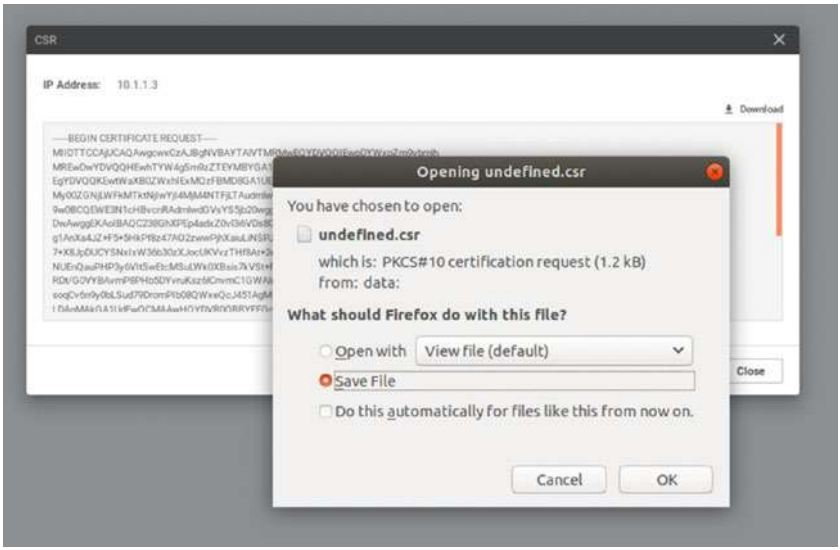

Figure 5-7. *The vBond CSR*

Sign the certificate using the Linux server:

```
openssl x509 -req -in undefined.csr -CA CA.crt -CAkey CA.key
-CAcreateserial -out vBond01.pem -days 1000 -sha256
```

Back on vManage, navigate to *Configuration* ➤ *Certificates* ➤ *Controllers*. Click "Install Certificate." Select the vBond01.pem certificate and install it.

The task view will first show that the install is scheduled (Figure 5-8).

Figure 5-8. *vBond certificate install scheduled*

Then, it will show that it was (hopefully) successful (Figure 5-9).

Figure 5-9. *vBond certificate install success*

Head back to *Configuration* ➤ *Devices* ➤ *Controllers*; we will see that the Certificate Status now shows as "Installed," and we should be fully populated with the hostname and system IP (Figure 5-10).

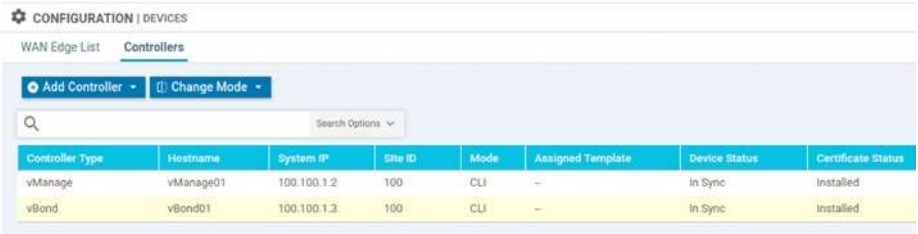

Figure 5-10. *All the vBond details!*

If you don't get the same result, then head over to the troubleshooting chapter (Chapter 14).

Assuming all is well, we should see that our dashboard has updated to reflect the new vBond orchestrator (Figure 5-11).

Figure 5-11. *We have a vBond*

Our Network page (Monitor ➤ Network) will also show the new controller (Figure 5-12).

Figure 5-12. *The Network monitoring page*

We can use the console to check our orchestrator properties ("`show orchestrator local-properties`"), which is great for troubleshooting certificate issues, as well as misconfiguration such as organization names that do not match.

```
vBond01#  show orchestrator local-properties
personality                    vbond
sp-organization-name           Learning_SD-WAN
organization-name              Learning_SD-WAN
system-ip                      100.100.1.3
certificate-status             Installed
root-ca-chain-status           Installed

certificate-validity           Valid
certificate-not-valid-before Apr 01 08:23:30 2020 GMT
certificate-not-valid-after  Dec 27 08:23:30 2022 GMT
chassis-num/unique-id          87563514-cc06-47a9-85ea-
                               03d4cf2e0a24
serial-num                     C024D682372213FA
number-active-wan-interfaces 1
protocol                       dtls

INSTANCE INDEX  PORT  VSMARTS  VMANAGES  STATE
-----------------------------------------------
0        0      12346 0        4         up

vBond01#
```

We can also check out the connections we have using the command "`show orchestrator connections`". The output for this is quite long, as shown in Figure 5-13!

Figure 5-13. *The output from "show orchestrator connections"*

By now, the serial number list we downloaded from SmartNet and uploaded to vManage should have downloaded to vBond (this output has been truncated to avoid sharing serial numbers):

```
vBond01# show orchestrator valid-vedges serial-number

CHASSIS NUMBER                SERIAL NUMBER
------------------------------------------------------------
00B6B5DD-15FB-123FFFF         1ddb68eb80270123FFFF
567F7A4B-0720-123FFFF         ceb5021fb63bd123FFFF
60389EF5-8D88-123FFFF         1ef3a68c75c43123FFFF
B0EA9F50-99AC-123FFFF         a2d1493536d8f123FFFF
CD24A6C9-330E-123FFFF         2664fe69a308c123FFFF
CSR-0502AB1A-123FFFF          2501e42780753123FFFF
CSR-17B237B8-123FFFF          e8c8481cc8dda123FFFF
CSR-E5C01068-123FFFF          26f0b73247dd4123FFFF
CSR-ED477D6F-123FFFF          a4f4d0216a7cb123FFFF
CSR-F1C7D091-123FFFF          7a652bb5c0b47123FFFF

vBond01#
```

The final stage of setting up the vBond controller is to enable the tunnel interface under VPN 0. We need this for the vSmart and edge devices to be able to connect to vBond.

```
vBond01# config
Entering configuration mode terminal
vBond01(config)# vpn 0
vBond01(config-vpn-0)# interface ge0/0
vBond01(config-interface-ge0/0)# tunnel-interface
vBond01(config-tunnel-interface)# allow-service all
vBond01(config-tunnel-interface)# encapsulation ipsec
vBond01(config-tunnel-interface)# commit and-quit
Commit complete.
vBond01#
```

Alternative vBond Deployments

We can also set up the vBond appliance on VMWare and KVM.

VMWare

Within vSphere, click *File* ➤ *Deploy OVF Template…*, then select viptela-edge-19.3.0-genericx86-64.ova, and click Next (Figure 5-14).

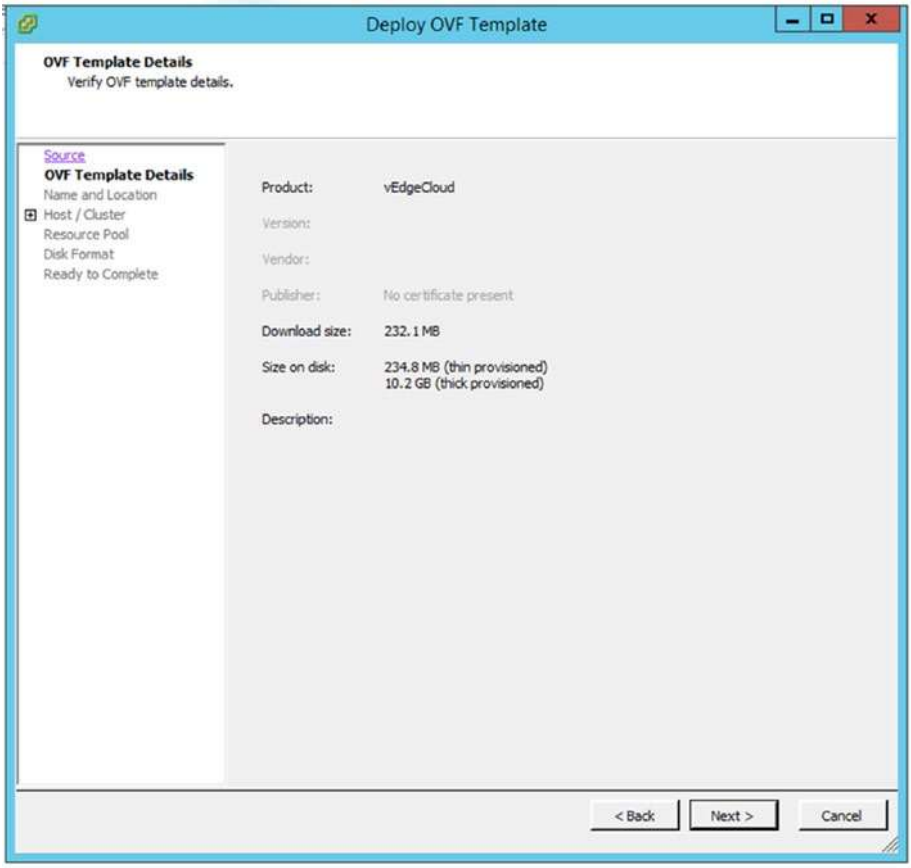

Figure 5-14. *vBond install on ESXi*

Click Next.

Name the new VM, and select a location (Figure 5-15).

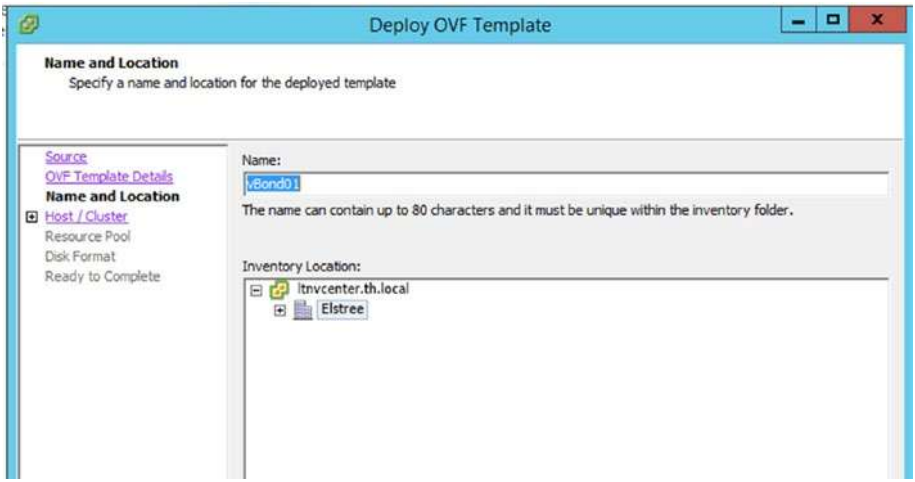

Figure 5-15. *Naming the vBond controller on ESXi*

Select a host and click Next.

Select a datastore.

Click next on Disk Format.

Select the networks you will be using; each should be mapped to different networks (ideally) (Figure 5-16).

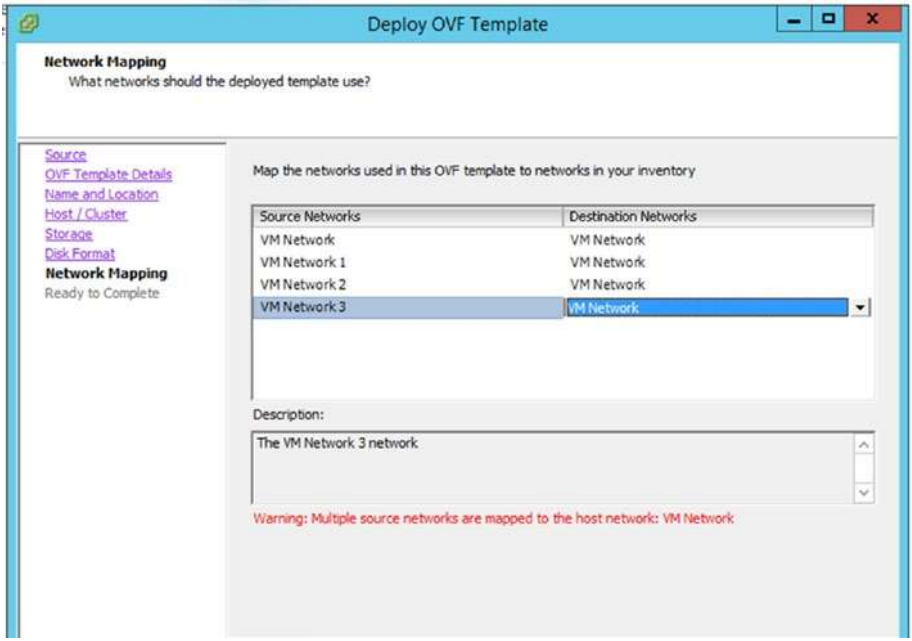

Figure 5-16. *vBond networks*

Click Next to complete the deployment settings, and finally, click Finish (Figure 5-17).

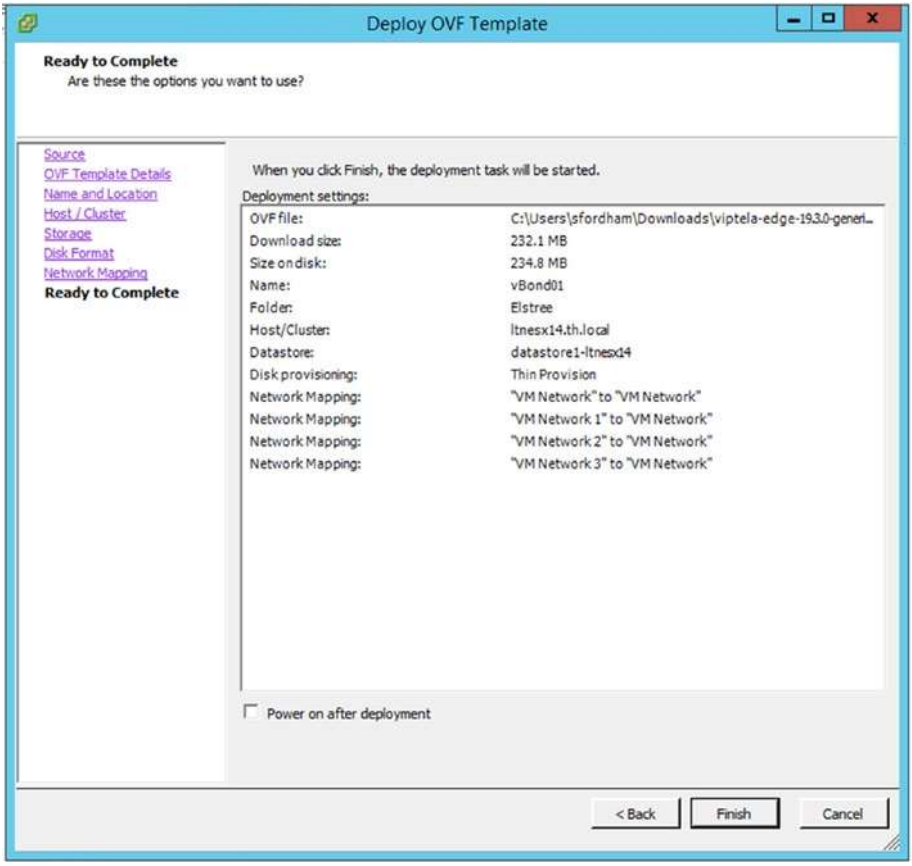

Figure 5-17. *vBond configuration*

The deployment will complete (Figure 5-18).

Figure 5-18. *vBond successful deployment*

KVM

If you want to install vBond on KVM, use the following:

```
virt-install \
    --name vbond01 \
    --os-type linux \
    --os-variant ubuntu14.04 \
    --cpu host \
    --vcpus=2 \
    --hvm \
    --arch=x86_64 \
    --ram 2048 \
    --disk path=viptela-bond-19.3.0-genericx86-64.qcow2,
      size=16,device=disk,bus=ide,format=qcow2 \
    --network=network:default,model=virtio \
    --network=network:default,model=virtio \
    --graphics none \
    --import
```

Summary

In this chapter, we have set up our vBond controller. The next step in building our topology is to set up vSmart.

CHAPTER 6

Deploying vSmart

The vSmart controller sits between the vEdge devices and the vBond orchestrator; it helps with the authentication of edge devices to the vBond and manages the connectivity between the edge devices.

vSmart Basic Config

The vSmart configuration follows the same steps as the vBond from the previous chapter:

```
vsmart# config
Entering configuration mode terminal
vsmart(config)# system
vsmart(config-system)# host-name vSmart01
vsmart(config-system)# organization-name Learning_SD-WAN
vsmart(config-system)# site-id 100
vsmart(config-system)# system-ip 100.100.1.4
vsmart(config-system)#
vsmart(config-system)# vbond 10.1.1.3
vsmart(config-system)#
```

For the VPN 0 configuration, we need to enable the NETCONF service in the tunnel interface:

```
vsmart(config-system)# vpn 0
vsmart(config-vpn-0)#
vsmart(config-vpn-0)# no interface eth0
vsmart(config-vpn-0)# interface eth1
vsmart(config-interface-eth1)# ip address 10.1.1.4/24
vsmart(config-interface-eth1)# no shut
vsmart(config-interface-eth1)# ip route 0.0.0.0/0 10.1.1.1
vsmart(config-interface-eth1)# tunnel-interface
vsmart(config-tunnel-interface)# allow-service netconf
vsmart(config-tunnel-interface)# commit and-quit
Commit complete.
vSmart01#
```

Whereas the vBond IP address should be a public IP (in a real-life deployment), the IP address we assign to the eth1 interface does not have to, but will be determined by your deployment needs. We can now test connectivity.

```
vSmart01# ping 10.1.1.2
Ping in VPN 0
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=3.41 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=1.48 ms
^C
--- 10.1.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 1.484/2.450/3.416/0.966 ms
vSmart01#
```

Let's set up our management connectivity:

```
vSmart01# config
vSmart01(config)# vpn 512
vSmart01(config-vpn-512)# interface eth0
vSmart01(config-interface-eth0)# ip address 100.1.1.4/24
vSmart01(config-interface-eth0)# no shutdown
vSmart01(config-interface-eth0)# exit
vSmart01(config-vpn-512)# commit and-quit
Commit complete.
vSmart01#
```

vSmart Certificates

This is the same process as the other devices. We start by copying the CA certificate from the Linux server to the vSmart controller.

```
scp CA.crt admin@10.1.1.4:
```

Then we install the certificate:

```
vSmart01# request root-cert-chain install /home/admin/CA.crt
Uploading root-ca-cert-chain via VPN 0
Copying ... /home/admin/CA.crt via VPN 0
Updating the root certificate chain..
Successfully installed the root certificate chain
vSmart01#
```

The next step is to add vSmart to vManage. Navigate to *Configuration ➤ Devices Controllers*, click "*Add Device*," and select vSmart. Add the IP address, username, and password, and set the protocol, which can be left as the default of "DTLS" (Figure 6-1).

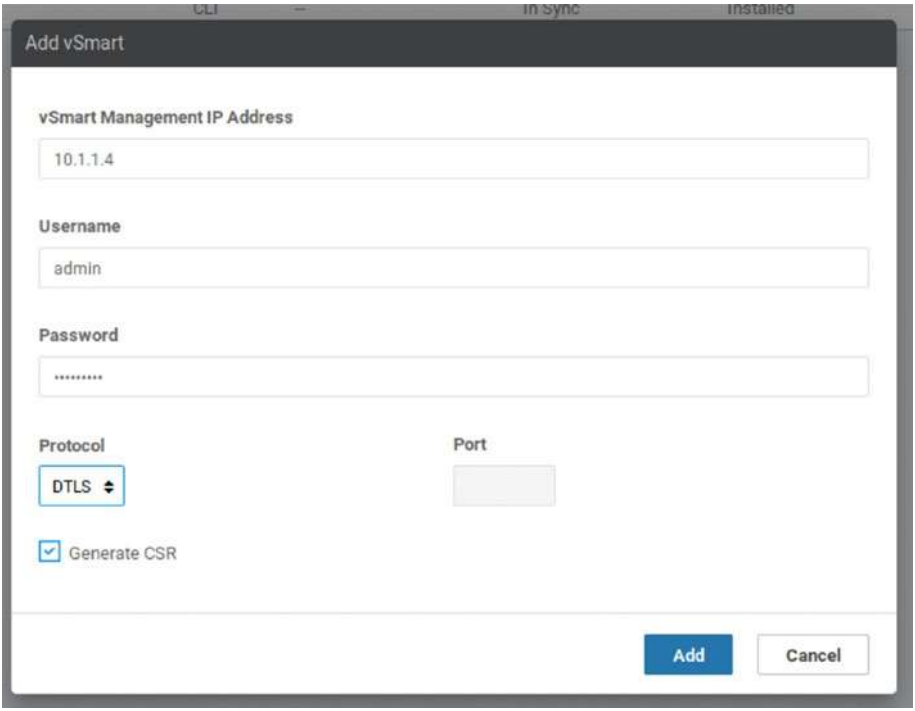

Figure 6-1. *Adding a vSmart controller*

View and save the CSR (Figure 6-2).

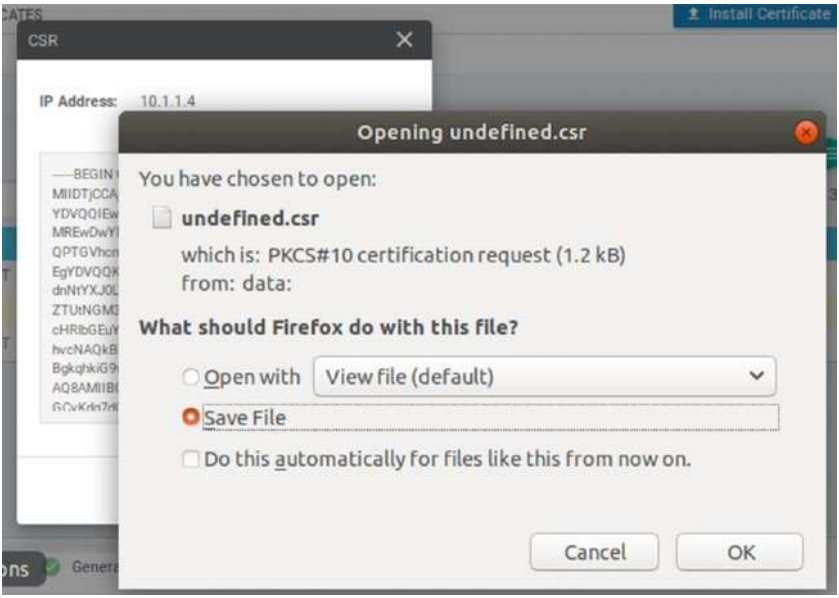

Figure 6-2. *The vSmart CSR*

Because we will already have a file called "undefined.csr", we should rename this new file when we generate the certificate (Figure 6-3).

```
user@user-virtual-machine:~/Downloads$ mv undefined\(1\).csr vSmart.csr
user@user-virtual-machine:~/Downloads$ openssl x509 -req -in vSmart.csr -CA CA.
crt -CAkey CA.key -CAcreateserial -out vSmart01.pem -days 1000 -sha256
Signature ok
subject=/C=US/ST=California/L=San Jose/OU=Learning_SD-WAN/O=Viptela LLC/CN=vsma
rt-611a5b84-b2e5-4c7f-951b-a91c93520063-0.viptela.com/emailAddress=support@vipt
ela.com
Getting CA Private Key
user@user-virtual-machine:~/Downloads$
```

Figure 6-3. *Generating the vSmart certificate*

Upload the certificate back into vManage, by going to *Configuration* ➤ *Certificates* ➤ *Controllers*, clicking "*Upload certificate*," and selecting the vSmart01.pem certificate we just created.

The vSmart should then be fully populated with the hostname and system IP (Figure 6-4).

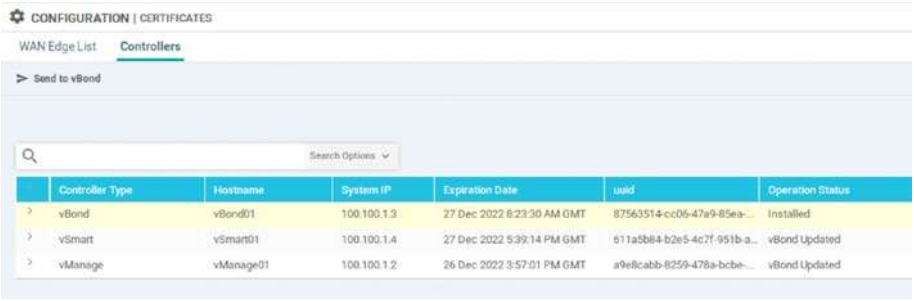

Figure 6-4. *vSmart showing in the NMS*

Similarly, the Devices page should show also be updated (Figure 6-5).

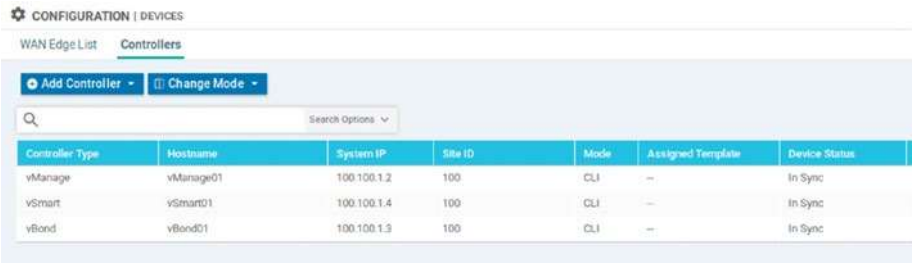

Figure 6-5. *vSmart in Devices*

If you are missing the hostname and system IP, then head over to the troubleshooting chapter.

Our dashboard should now show that we have one vSmart (Figure 6-6).

Figure 6-6. *vSmart in the dashboard*

This should be reflected in the Network page (Figure 6-7).

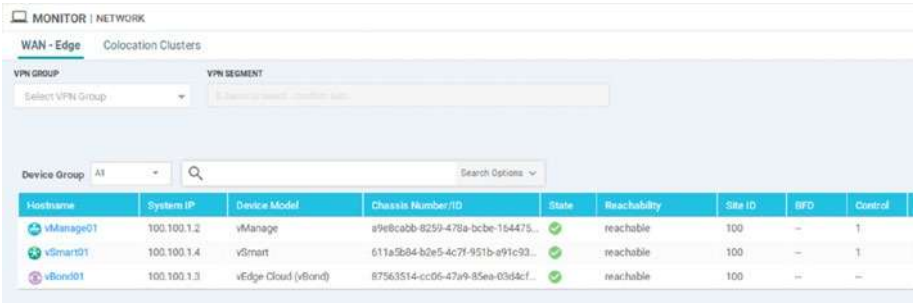

Figure 6-7. *The Network page*

We can also use the CLI to gauge the health of our vSmart controller, by checking the connections to our vBond device, using the command "`show transport connection`":

```
vSmart01# show transport connection

TRACK
TYPE     SOURCE   DESTINATION   HOST   INDEX   TIME            STATE
-----------------------------------------------------------------
system      -     10.1.1.3            0       Wed Apr 1 20    up

vSmart01#
```

As well as looking at this (truncated) output from "`show control local-properties`":

```
vSmart01# show control local-properties
personality                          vsmart
sp-organization-name                 Learning_SD-WAN
organization-name                    Learning_SD-WAN
root-ca-chain-status                 Installed
```

```
certificate-status                Installed
certificate-validity              Valid
certificate-not-valid-before      Apr 01 17:39:14 2020 GMT
certificate-not-valid-after       Dec 27 17:39:14 2022 GMT

dns-name                          10.1.1.3
site-id                           100
domain-id                         1
protocol                          dtls
tls-port                          23456
system-ip                         100.100.1.4
chassis-num/unique-id             611a5b84-b2e5-4c7f-951b-
a91c93520063
serial-num                        C024D682372213FB
token                             -NA-
retry-interval                    0:00:00:17
no-activity-exp-interval          0:00:00:20
dns-cache-ttl                     0:00:02:00
port-hopped                       FALSE
time-since-last-port-hop          0:00:00:00
cdb-locked                        false
number-vbond-peers                1

INDEX   IP                                        PORT
-----------------------------------------------------
0       10.1.1.3                                  12346

number-active-wan-interfaces      2

vSmart01#
```

The output has been truncated due to the connection details at the end, which have been removed to preserve print formatting. From here, we can check our certificate validity.

We can also check our control connections using the command "`show control connections`" (Figure 6-8).

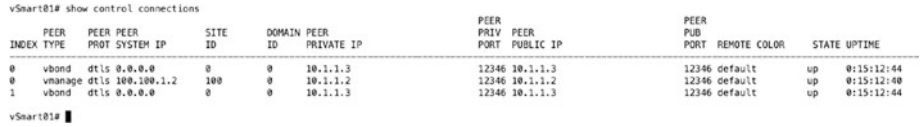

Figure 6-8. *Show control connections*

We have full connections to the vBond and vManage over DTLS.

vSmart Authentication and Validation

Behind the scenes, the vSmart controller and the vBond start a mutual process of validation and authentication between each other. We start this process when we add the IP address (or DNS name) of the vBond device during the initial configuration of the vSmart device:

```
vsmart(config-system)# vbond 10.1.1.3
```

The second part of this initial validation and authentication is performed when we add vSmart to vManage and issue the certificates. vManage then initiates an update of vBond (Figure 6-9).

Figure 6-9. *vManage updating vBond with the details of vSmart*

vManage has now sent the serial number of vSmart to vBond, priming it for vSmart to start talking to it over a DTLS tunnel, which is encrypted using RSA private- and public-key pairs.

With the tunnel in place, vBond sends the root CA to vSmart, along with the vEdge serial number file. vSmart checks the organization name in the certificate against its own configured organization name. If the organization name is the same, then vSmart will check the certificate against the CA root chain (which explains why we have uploaded the CA.crt file every time we have started a new device up). If the certificate signature is correct, the DTLS tunnel stays up and the authentication of the vBond orchestrator to vSmart has completed.

Naturally, vBond must perform similar checks. So vSmart also sends the root CA certificate to vBond, and vBond checks the serial number, organization name, and certificate signature as well. If these checks pass, then authentication of vSmart to vBond has completed.

The temporary DTLS tunnel that was created for this initial authentication now transitions to a permanent tunnel.

If the vBond orchestrator has not started when the vSmart controller is in the position to start the authentication sequence, then the vSmart will periodically attempt to start the sequence, until it is eventually successful.

If we have more than one vSmart controller, then the process is much the same. Each vSmart controller learns of the other vSmart controller(s) from vBond. Once each vSmart controller receives the serial number file from the vBond orchestrator, it initiates a DTLS connection to the other vSmart controller. The first vSmart will send its trusted root CA-signed certificate to the other vSmart controller. The second vSmart controller checks the serial number of the first vSmart controller against the serial number file it has received from vBond and checks the organization name in the certificate and the signature of the certificate. This process is also performed by the other vSmart controller so that there is a two-way trust between the controllers. Again, if each step is completed successfully, the

temporary DTLS tunnel is replaced with a permanent one. vBond will then balance the control connections between the vEdge devices and the vSmart controllers automatically.

If any of the steps fail, and the same is true for the authentication sequence between vSmart and vBond devices, then the temporary DTLS tunnel will be torn down and the authentication attempt stops.

Alternative vSmart Deployments

Like the other appliances, we can run the vSmart on VMWare or KVM.

VMWare

As the steps for installing the vSmart are no different than those of vManage or vBond, there is no benefit in showing you all the steps.

KVM

From the Linux command prompt type, the following:

```
virt-install \
     --name vSmart01 \
     --os-type linux \
     --os-variant ubuntu14.04 \
     --cpu host \
     --vcpus=2 \
     --hvm \
     --arch=x86_64 \
     --ram=2048 \
     --disk path=viptela-smart-19.3.0-genericx86_64.qcow2,
      size=16,device=disk,bus=ide,format=qcow2 \
```

```
--network=network:default,model=virtio \
--network=network:default,model=virtio \
--graphics none \
--import
```

Summary

We have now set up our vSmart controller which acts as the gateway between the vBond and vManage devices and the vEdge devices. We will look at the role of the vSmart controller further, when we start to provision policies in Chapter 10, but for now, we are at a position where we can start adding our vEdge devices.

CHAPTER 7

Edge Devices

In this chapter, we will set up two vEdge devices and one cEdge device. vEdge means that it is a Viptela device, and cEdge means that it is a Cisco device (a CSR1000v in this instance). Although they are technically now the same company, vEdge and cEdge are still used to distinguish between the two different router types and the OS that they run.

We begin with the vEdge router, in our HQ, in the same way as the other devices we have previously configured, by changing the password:

```
vedge login: admin
Password:
Welcome to Viptela CLI
admin connected from 127.0.0.1 using console on vedge
You must set an initial admin password.
Password:
Re-enter password:
vedge#
```

The basic set up is the same as the other devices we have configured:

```
vedge# config
Entering configuration mode terminal
vedge(config)# system
vedge(config-system)# host-name vEdge01
vedge(config-system)# site-id 100
vedge(config-system)# system-ip 100.100.1.5
vedge(config-system)# vbond 10.1.1.3
```

```
vedge(config-system)# organization-name Learning_SD-WAN
vedge(config-system)#
vedge(config-system)# commit and-quit
Commit complete.
vEdge01#
```

We can then move on to VPN 0:

```
vEdge01(config)# vpn 0
vEdge01(config-vpn-0)# ip route 0.0.0.0/0 10.1.1.1
vEdge01(config-vpn-0)# interface ge0/0
vEdge01(config-interface-ge0/0)# ip address 10.1.1.5/24
vEdge01(config-interface-ge0/0)# no shut
vEdge01(config-interface-ge0/0)#
vEdge01(config-interface-ge0/0)# tunnel-interface
vEdge01(config-tunnel-interface)# encapsulation ipsec
vEdge01(config-tunnel-interface)# color blue
vEdge01(config-tunnel-interface)#
vEdge01(config-tunnel-interface)# validate
Validation complete
vEdge01(config-tunnel-interface)# commit and-quit
Commit complete.
vEdge01#
```

Once the changes have been committed, confirm that you can reach the vBond and vSmart devices:

```
vEdge01# ping 10.1.1.3
Ping in VPN 0
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=64 time=26.2 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=64 time=28.8 ms
^C
```

```
--- 10.1.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 26.239/27.525/28.811/1.286 ms
vEdge01#
vEdge01# ping 10.1.1.4
Ping in VPN 0
PING 10.1.1.4 (10.1.1.4) 56(84) bytes of data.
64 bytes from 10.1.1.4: icmp_seq=1 ttl=64 time=1.36 ms
64 bytes from 10.1.1.4: icmp_seq=2 ttl=64 time=1.31 ms
^C
--- 10.1.1.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.316/1.340/1.365/0.044 ms
vEdge01#
```

Next, on the vManage server, navigate to *Configuration* ➤ *Devices* ➤ *WAN Edge List*. Select the first vEdge Cloud device (not the CSR1000v), and click the three dots on the right-hand side, and select "Generate Bootstrap Configuration" from the drop-down list (Figure 7-1).

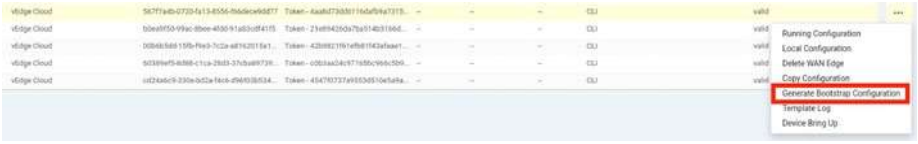

Figure 7-1. *Generating the vEdge bootstrap configuration*

We need the configuration to be in "Cloud-Init" format. Click OK (Figure 7-2).

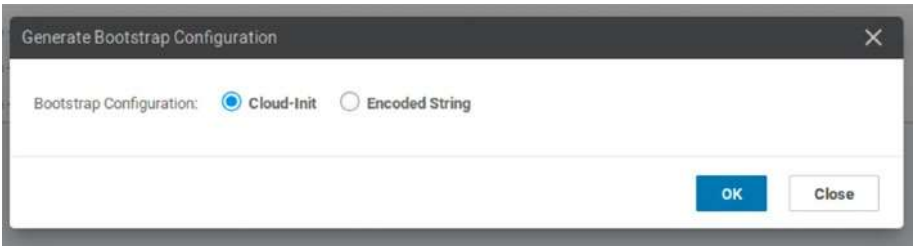

Figure 7-2. *Select Cloud-Init*

The file contains a few lines of text as well as the certificates we need for the communication to occur between the edge device and the control devices (Figure 7-3).

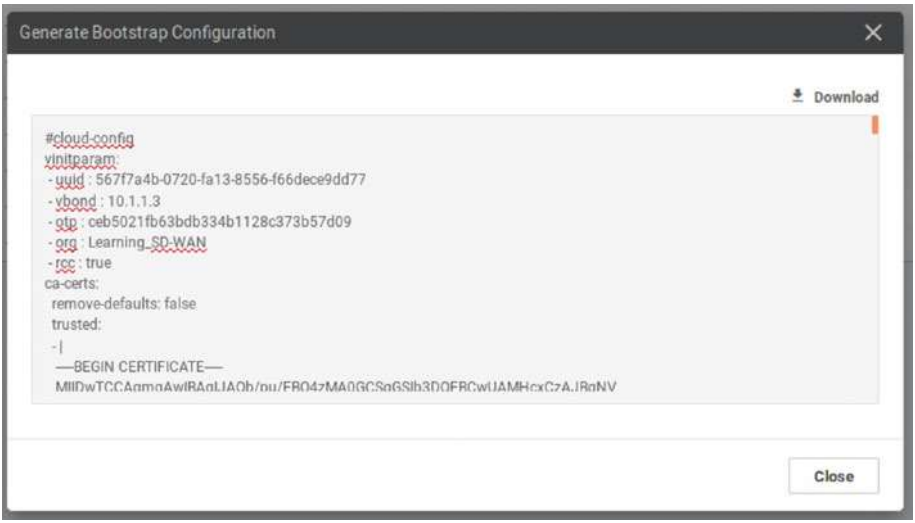

Figure 7-3. *The vEdge bootstrap configuration*

Save the file locally, and open it up in a text editor, so that we can copy the parts we need, which are the UUID and the OTP (one-time password). The UUID is our chassis number, and the OTP is the one-time token code. The OTP is referred to as a "token" in the CLI and as an "OTP" in the vManage GUI, so these terms may be used interchangeably.

Back on the vEdge router, request activation using the "`request vedge-cloud activate chassis-number <UUID> token <OTP>`" command using the UUID and the OTP from the file we just downloaded:

```
vEdge01# request vedge-cloud activate chassis-number 577f7a4b-
0720-fa13-8556-f66dece9dd77 token ceb5021fb63bdb334b1128c373b5
7d09
```

As we are using an OTP, the command can only be run once. If the activation fails, then the device must be decommissioned from vManage, so that the serial is released for reuse and a new OTP must be generated. The UUID will remain the same though.

For the certificate chain to be valid, we need to upload the CA certificate from the Linux server. Start by enabling SSH on the vEdge device.

```
vEdge01# config
Entering configuration mode terminal
vEdge01(config)# vpn 0
vEdge01(config-vpn-0)# interface ge0/0
vEdge01(config-interface-ge0/0)# tunnel-interface
vEdge01(config-tunnel-interface)# allow-service sshd
vEdge01(config-tunnel-interface)#
vEdge01(config-tunnel-interface)# commit and-quit
Commit complete.
vEdge01#
```

Upload the certificate from the Linux device:

```
scp CA.crt admin@10.1.1.5:
```

Then install the CA certificate:

```
vEdge01# request root-cert-chain install /home/admin/CA.crt
Uploading root-ca-cert-chain via VPN 0
Copying ... /home/admin/CA.crt via VPN 0
Updating the root certificate chain..
Successfully installed the root certificate chain
vEdge01#
```

Assuming all goes well, we should see the edge device listed in vManage (Figure 7-4).

Figure 7-4. *Our edge device showing in vManage*

If we look at our network now (*Monitor* ➤ *Network*), we should see the edge device (Figure 7-5).

Hostname	System IP	Device Model	Chassis Number/ID	State	Reachability	Site ID	BFD	Control
vManage01	100.100.1.2	vManage	713d0c64-cd24-40f8-b7...		reachable	100	–	2
vSmart01	100.100.1.4	vSmart	5931fdfa-6406-4c85-b6e...		reachable	100	–	2
vBond01	100.100.1.3	vEdge Cloud (vBond)	0b4e8edb-754b-4bf6-ac...		reachable	100	–	–
vEdge01	100.100.1.5	vEdge Cloud	567f7a4b-0720-fa13-855...		reachable	100	0	2

Figure 7-5. *The network monitoring page*

It will also be visible in our WAN Edge List (Figure 7-6).

State	Device Model	Chassis Number	Serial No./Token	Hostname	System IP	Site ID	Mode	Device Status	Validity
	CSR1000v	CSR-0502AB1A-7DED-13AE-3AB1-776B...	Token - 553f67a845b6d4fe4a443e1c43e9...	–	–	–	CLI		valid
	CSR1000v	CSR-178237B8-852A-7186-3C94-27361...	Token - e04cf06c0bb69ebce6971c42911...	–	–	–	CLI		valid
	CSR1000v	CSR-ED477D6F-238C-1B08-0857-E19E...	Token - b7681ad8e388667eeb2117cee57...	–	–	–	CLI		valid
	CSR1000v	CSR-F1C7D091-3AF3-4A45-4C9B-B31F...	Token - 86ac8738253a5e55074de6f5c59...	–	–	–	CLI		valid
	CSR1000v	CSR-E5C01068-E2C8-D2F7-AE80-3D32...	Token - 6d04f5ffc42814e21b507c6f4a74...	–	–	–	CLI		valid
	vEdge Cloud	567f7a4b-0720-fa13-8556-f66dece9dd77	68EFF562	vEdge01	100.100.1.5	100	CLI	In Sync	valid
	vEdge Cloud	b0ea9f50-99ac-8bee-4fdd-91a83cdf4115	Token - 21e89426da7ba514b3166d85490...	–	–	–	CLI		valid

Figure 7-6. *The edge list*

We can look at the vSmart controller and see the OMP connections:

```
vSmart01# show omp summary
oper-state              UP
admin-state             UP
personality             vsmart
omp-uptime              4:20:06:20
routes-received         0
routes-installed        0
routes-sent             0
tlocs-received          1
tlocs-installed         1
tlocs-sent              0
services-received       0
services-installed      0
services-sent           0
mcast-routes-received   0
mcast-routes-installed  0
mcast-routes-sent       0
hello-sent              5
hello-received          6
handshake-sent          1
handshake-received      1
alert-sent              0
alert-received          0
inform-sent             3
inform-received         3
update-sent             0
update-received         1
policy-sent             0
policy-received         0
total-packets-sent      9
```

```
total-packets-received 11
vsmart-peers           0
vedge-peers            1
vSmart01#
```

Note that we have received (and installed) a TLOC, but have not received any routes yet. So, let's add a route on our edge router and see what happens. This will become VPN 1. We can support a huge amount of VPNs, from 1 to 511 and then from 513 to 65527! That said, the vEdge devices can support a total of 64 concurrent VPNs.[1]

```
vEdge01# config
Entering configuration mode terminal
vEdge01(config)# vpn ?
This line doesn't have a valid range expression
Possible completions:
  Allowed values on vedge: <0..65527>
  Allowed values on vsmart/vmanage/vcontainer: <0 and 512>
  0
  512
vEdge01(config)#
vEdge01(config)# vpn 1
vEdge01(config-vpn-1)# interface loopback1
vEdge01(config-interface-loopback1)# ip address 172.16.10.1/24
vEdge01(config-interface-loopback1)# no shut
vEdge01(config-interface-loopback1)# end
Uncommitted changes found, commit them? [yes/no/CANCEL] yes
Commit complete.
vEdge01#
```

[1] www.slideshare.net/CiscoCanada/understanding-cisco-next-generation-sdwan-solution

Now if we look at the vSmart controller again, we can see that we have received an additional route:

```
vSmart01# show omp summary
oper-state              UP
admin-state             UP
personality             vsmart
omp-uptime              4:20:59:09
routes-received         1
routes-installed        0
routes-sent             0
tlocs-received          1
tlocs-installed         1
tlocs-sent              0
services-received       1
services-installed      1
services-sent           0
mcast-routes-received   0
mcast-routes-installed  0
mcast-routes-sent       0
hello-sent              164
hello-received          166
handshake-sent          1
handshake-received      1
alert-sent              0
alert-received          0
inform-sent             7
inform-received         7
update-sent             0
update-received         3
policy-sent             0
policy-received         0
```

```
total-packets-sent     172
total-packets-received 177
vsmart-peers           0
vedge-peers            1
vSmart01#
```

We can use the "`show omp routes`" command to look at the routes we have received. Note that the tloc color (blue) matches what we configured on the vEdge01 device when we set it up:

```
vSmart01# show omp routes

------------------------------------------------
omp route entries for vpn 1 route 172.16.10.0/24
------------------------------------------------
            RECEIVED FROM:
peer            100.100.1.5
path-id         74
label           1003
status          C,R
loss-reason     not set
lost-to-peer    not set
lost-to-path-id not set
    Attributes:
     originator       100.100.1.5
     type             installed
     tloc             100.100.1.5, blue, ipsec
     ultimate-tloc    not set
     domain-id        not set
     overlay-id        1
     site-id          100
     preference       not set
```

```
     tag               not set
     origin-proto      connected
     origin-metric     0
     as-path           not set
     unknown-attr-len not set
vSmart01#
```

Let's add vEdge02, following the configuration given here:

```
vedge# config
Entering configuration mode terminal
vedge(config)# system
vedge(config-system)# host-name vEdge02
vedge(config-system)# site-id 100
vedge(config-system)# system-ip 60.100.100.1
vedge(config-system)# vbond 10.1.1.3
vedge(config-system)# organization-name Learning_SD-WAN
vedge(config-system)# vpn 0
vedge(config-vpn-0)# no interface ge0/0
vedge(config-vpn-0)# ip route 0.0.0.0/0 50.11.11.254
vedge(config-vpn-0)# interface ge0/1
vedge(config-interface-ge0/1)# ip address 50.11.11.1/24
vedge(config-interface-ge0/1)# no shut
vedge(config-interface-ge0/1)# tunnel-interface
vedge(config-tunnel-interface)# encapsulation ipsec
vEdge02(config-tunnel-interface)# allow-service sshd
vedge(config-tunnel-interface)# color red
vedge(config-tunnel-interface)# commit and-quit
Commit complete.
vEdge02#
```

Note I have placed it into site 100, instead of site 200 as shown in the topology in Chapter 2. This is on purpose and will be explained in the next chapter.

Before moving on, we should check our connectivity, first by pinging ISP-R, and then vBond:

```
vEdge02# ping 50.11.11.254
Ping in VPN 0
PING 50.11.11.254 (50.11.11.254) 56(84) bytes of data.
64 bytes from 50.11.11.254: icmp_seq=1 ttl=64 time=0.900 ms
64 bytes from 50.11.11.254: icmp_seq=2 ttl=64 time=0.521 ms
^C
--- 50.11.11.254 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.521/0.710/0.900/0.191 ms
vEdge02#
vEdge02# ping 10.1.1.3
Ping in VPN 0
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=62 time=25.8 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=62 time=27.0 ms
^C
--- 10.1.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 25.814/26.456/27.099/0.662 ms
vEdge02#
```

Next, we can copy the CA.crt over to vEdge02 (from the Linux server):

```
scp CA.crt admin@50.11.11.1:
```

Then, install the certificate:

```
vEdge02# request root-cert-chain install /home/admin/CA.crt
Uploading root-ca-cert-chain via VPN 0
Copying ... /home/admin/CA.crt via VPN 0
Updating the root certificate chain..
Successfully installed the root certificate chain
vEdge02#
```

On vManage, we can navigate to *Configuration* ➤ *Devices* and generate the bootstrap code for the second vEdge Cloud device on the list and activate it:

```
vEdge02# request vedge-cloud activate chassis-
number b0ea9f50-99ac-8bee-4fdd-91a83cdf41f5 token
a2d1493536d8f4b2aebdc7f845d416
vEdge02#
```

We should now see that the device list has been updated (Figure 7-7).

State	Device Model	Chassis Number	Serial No./Token	Hostname	System IP	Site ID	Mode	Device Status
	CSR1000v	CSR-0502AB1A-7DED-...	Token - 553f67a845b6...	–	–	–	CLI	
	CSR1000v	CSR-17B237B8-852A-7...	Token - e04cf06c0bb69...	–	–	–	CLI	
	CSR1000v	CSR-ED477D6F-238C-1...	Token - b7681adde388...	–	–	–	CLI	
	CSR1000v	CSR-F1C7D091-3AF3-4...	Token - 86ac8738253a...	–	–	–	CLI	
	CSR1000v	CSR-E5C01068-E2C8-0...	Token - 0d04f5ffc4281...	–	–	–	CLI	
	vEdge Cloud	567f7a4b-0720-fa13-8...	68EFF562	vEdge01	100.100.1.5	100	CLI	In Sync
	vEdge Cloud	b0ea9f50-99ac-8bee-4...	5A096571	vEdge02	60.100.100.1	100	CLI	In Sync
	vEdge Cloud	00b66b5dd-15fb-f9e3-7...	Token - 42b9821f61efb...	–	–	–	CLI	

Figure 7-7. *The WAN edge device list*

Similarly, our list of network devices has also been updated (Figure 7-8).

Hostname	System IP	Device Model	Chassis Number/ID	State	Reachability	Site ID	BFD	Control
vManage01	100.100.1.2	vManage	713d0c64-cd24-40f8-b7...		reachable	100	–	3
vSmart01	100.100.1.4	vSmart	5931fdfa-6406-4c85-b6e...		reachable	100	–	3
vBond01	100.100.1.3	vEdge Cloud (vBond)	0b4e8edb-754b-4bf6-ac...		reachable	100	–	--
vEdge01	100.100.1.5	vEdge Cloud	567f7a4b-0720-fa13-855...		reachable	100	0	2
vEdge02	60.100.100.1	vEdge Cloud	b0ea9f50-99ac-8bee-4fd...		reachable	100	0	2

Figure 7-8. *The Network monitor window (again)*

Let's test OMP by creating a new VPN on vEdge02 (as we did on vEdge01):

```
vEdge02# config
Entering configuration mode terminal
vEdge02(config)# vpn 1
vEdge02(config-vpn-1)# interface loopback1
vEdge02(config-interface-loopback1)# ip address 172.16.20.1/24
vEdge02(config-interface-loopback1)# no shutdown
vEdge02(config-interface-loopback1)# end
Uncommitted changes found, commit them? [yes/no/CANCEL] yes
Commit complete.
vEdge02#
```

We can confirm that we can see both of the routes using the command "show omp routes received":

```
vEdge02# show omp routes received

-------------------------------------------------
omp route entries for vpn 1 route 172.16.10.0/24
-------------------------------------------------
            RECEIVED FROM:
peer            100.100.1.4
path-id         1
label           1003
status          Inv,U
loss-reason     not set
lost-to-peer    not set
lost-to-path-id not set
    Attributes:
     originator       100.100.1.5
     type             installed
     tloc             100.100.1.5, blue, ipsec
     ultimate-tloc    not set
```

```
     domain-id        not set
     overlay-id        1
     site-id          100
     preference       not set
     tag              not set
     origin-proto     connected
     origin-metric    0
     as-path          not set
     unknown-attr-len not set

------------------------------------------------
omp route entries for vpn 1 route 172.16.20.0/24
------------------------------------------------
            RECEIVED FROM:
peer            0.0.0.0
path-id         72
label           1003
status          C,Red,R
loss-reason     not set
lost-to-peer    not set
lost-to-path-id not set
    Attributes:
     originator       60.100.100.1
     type             installed
     tloc             60.100.100.1, red, ipsec
     ultimate-tloc    not set
     domain-id        not set
     overlay-id        1
     site-id          100
     preference       not set
     tag              not set
```

```
     origin-proto     connected
     origin-metric    0
     as-path          not set
     unknown-attr-len not set
vEdge02#
```

So far, so good. In the final part of our topology configuration, we will set up the CSR1000V.

CSR1000v

The commands used by the CSR1000v differ (slightly) from the other configurations we have encountered so far, as you might expect coming from a different vendor.

We start as usual, by logging in using admin/admin as the username and password and then setting a new password when prompted:

```
User Access Verification

Username: admin
Password:

Default admin password needs to be changed.

Enter new password:
Confirm password:
System status solid green (reason: All daemons up)
Router>
Router>
Successfully set new admin password
Router>
```

From here on, things get a bit different:

```
Router>en
Router#config
This command is not supported

Router#configure terminal
This command is not supported

Router#
```

We can no longer use the "config" command. Instead, we need to use the "config-transaction" command:

```
Router#config-transaction

admin connected from 127.0.0.1 using console on Router
Router(config)#
```

Now we are progressing. The hostname command is not configured under the system options (unlike the vEdges); instead, it is set at the top level of the configuration:

```
Router(config)# hostname CSR-1
```

The SD-WAN parameters are under the system command, though:

```
Router(config)# system
Router(config-system)# site-id 100
Router(config-system)# organization-name Learning_SD-WAN
Router(config-system)# vbond 10.1.1.3
Router(config-system)# system-ip 70.100.100.1
Router(config-system)#

Router(config-system)# commit
Commit complete.
CSR-1(config-system)#
```

The way we configure VPN 0 is also different:

```
CSR-1(config-system)# vpn 0
----------------------^
syntax error: unknown command
CSR-1(config-system)#
```

We have to jump around the console to get the SD-WAN tunnel prepared, starting by configuring the interface:

```
CSR-1(config-system)# exit
CSR-1(config)#
CSR-1(config)# interface GigabitEthernet 1
CSR-1(config-if)#
CSR-1(config-if)# ip address 50.10.10.1 255.255.255.0
CSR-1(config-if)# no shut
CSR-1(config-if)# ip route 0.0.0.0 0.0.0.0 50.10.10.254
CSR-1(config)# commit
Commit complete.
CSR-1(config)# end
```

We can test connectivity now to both our default gateway (ISP-R) and the vBond controller:

```
CSR-1#ping 50.10.10.254
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 50.10.10.254, timeout is 2
seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max =
1/5/22 ms
CSR-1#ping 10.1.1.3
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 10.1.1.3, timeout is 2
seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max =
20/27/30 ms
CSR-1#
```

Next, we create a tunnel interface, which is a hidden option, so make sure you don't put a space between "tunnel" and "1"; otherwise, the command will fail. Start with a "`show interface summary`" so you can copy and paste easily later on (I have truncated the output for formatting). We also need to set the tunnel mode to "sdwan."

```
CSR-1#show interface summary

Interface            IHQ    IQD  OHQ  OQD  RXBS  RXPS   TXBS
-------------------------------------------------------------
* GigabitEthernet1   0      0    0    0    0     0      0
* GigabitEthernet2   0      0    0    0    0     0      0
* GigabitEthernet3   0      0    0    0    0     0      0
* GigabitEthernet4   0      0    0    0    0     0      0
* Loopback65528      0      0    0    0    0     0      0
CSR-1#
CSR-1#config-transaction
CSR-1(config)# interface ?
Possible completions:
  ATM                    ATM-ACR
  AppGigabitEthernet     AppNav-Compress
  AppNav-UnCompress      BD-VIF
  BDI                    CEM
  CEM-ACR                Cellular
  Dialer                 Embedded-Service-Engine
  Ethernet               Ethernet-Internal
```

```
  FastEthernet           FiveGigabitEthernet
  FortyGigabitEthernet   GMPLS
  GigabitEthernet        Group-Async
  HundredGigE            LISP
  Loopback               Multilink
  Port-channel           SM
  Serial                 Service-Engine
  TenGigabitEthernet     Tunnel
  TwentyFiveGigE         TwentyFiveGigabitEthernet
  TwoGigabitEthernet     Vif
  Virtual-PPP            Virtual-Template
  VirtualPortGroup       Vlan
  Wlan-GigabitEthernet   nat64
  nat66                  nve
  ospfv3                 overlay
  pseudowire             ucse
  vasileft               vasiright
CSR-1(config)# interface Tunnel1
CSR-1(config-if)# ip unnumbered GigabitEthernet1
CSR-1(config-if)# tunnel source GigabitEthernet1
CSR-1(config-if)# tunnel mode sdwan
CSR-1(config-if)# no shutdown
CSR-1(config-if)#
CSR-1(config-if)# commit
Commit complete.
CSR-1(config-if)#
*Apr  7 19:35:34.704: %LINEPROTO-5-UPDOWN: Line protocol on
Interface Tunnel1, changed state to down
CSR-1(config-if)#
```

Next, we configure the SD-WAN commands, setting the physical interface, the encapsulation, and the color:

```
CSR-1(config)# sdwan
CSR-1(config-sdwan)# interface GigabitEthernet1
CSR-1(config-interface-GigabitEthernet1)# tunnel-interface
CSR-1(config-tunnel-interface)# encapsulation ipsec
CSR-1(config-tunnel-interface)# color biz-internet
CSR-1(config-tunnel-interface)# commit
Commit complete.
CSR-1(config-tunnel-interface)#
*Apr  7 19:40:45.596: %LINEPROTO-5-UPDOWN: Line protocol on
Interface Tunnel1, changed state to up
CSR-1(config-tunnel-interface)#
```

Whatever you do, don't be tempted to shorten the interface name to Gi1 (as most engineers would); the IOS XE CLI is *very* picky, and the interface name you use in the tunnel declaration and the SDWAN configuration has to match the output of "`show interface summary`".

Select the three dots next to the first CSR1000v device under *Configuration* ➤ *Devices*, and select the Generate Bootstrap Configuration option from the drop-down, selecting Cloud-Init when prompted. Save the file, and open it in a text editor (again so that we can grab the UUID and OTP).

Enable SSHD on the router, so that we can SCP the file across, and we might as well enable NETCONF, while we are there:

```
CSR-1(config-tunnel-interface)# allow-service sshd
CSR-1(config-tunnel-interface)# allow-service netconf
CSR-1(config-tunnel-interface)# commit
CSR-1(config-tunnel-interface)# end
```

Copy the certificate over from the Linux server, the command for which is slightly different for this device:

```
scp CA.crt admin@50.10.10.1:bootflash:/CA.crt
```

Check that the file has been copied over:

```
CSR-1#dir | i CA
   21  -rw-   1363   Apr 9 2020 08:56:49 +00:00  CA.crt
CSR-1#
```

SSH onto the router from the Linux machine, and then enter the following, copying and pasting the chassis number and OTP into the appropriate places:

```
CSR-1# request platform software sdwan vedge_cloud activate
chassis-number <uuid> token <otp>
```

Note how different the command is compared to the vEdge devices we set up previously:

```
vEdge02# request vedge-cloud activate chassis-number <uuid>
token <otp>
vEdge02#
```

With the cEdge, we need to install the new image and activate it; this is why we specify the "`platform software sdwan`".

Install the CA certificate, and watch the magic happen!

```
CSR-1#request platform software sdwan root-cert-chain install
bootflash:CA.crt
Uploading root-ca-cert-chain via VPN 0
Copying ... /bootflash/CA.crt via VPN 0
Updating the root certificate chain..
Successfully installed the root certificate chain
CSR-1#
```

We can now see the CSR-1 router under devices (Figure 7-9).

Figure 7-9. *Our CSR-1 device*

It has also increased the edge device count on our dashboard (Figure 7-10).

Figure 7-10. *Our updated dashboard*

Now we will turn our attention to what happens behind the scenes when we add an edge device onto the network.

vEdge Authentication

When we deploy vEdge routers, they need to connect to the vManage server so it may receive the configuration and to the vSmart controller so it can become part of the overlay network.

Before the vEdge router can talk to the vManage or vSmart devices, it builds a connection to the vBond, and once this has been completed, the vBond sends the IP addresses of vManage and vSmart to the vEdge.

The vEdge starts by initiating a DTLS connection to the IP address of the vBond device that is set in its configuration. This is secured through RSA private and public keys.

The vBond then works out whether the vEdge is behind a NAT device and, if it is, maps the public IP address and port to the private IP address. Now that the DTLS tunnel has been established, the two devices can authenticate against each other.

Firstly, vBond sends the root CA-signed certificate to the vEdge router. The organization name is extracted from the certificate, and vEdge compares the one in the certificate to the one locally configured. If the names match, then vEdge checks that the vBond certificate is signed by the root CA. If the organization name and the certificate pass the check, then vBond has authenticated to vEdge. If either of these two checks fails, the DTLS tunnel is torn down and periodically retried. You can see a Wireshark capture of this traffic in Chapter 14.

Now, vEdge must authenticate to vBond.

vBond starts by sending A 256-bit challenge (a random value), and in return, vEdge sends the following to vBond:

- The serial number
- The chassis number
- The board ID certificate (located on a chip inside the router)
- A 256-bit random value signed with the vEdge's private key

Once the vBond has received these, it compares the serial number and chassis number to the list it has been sent by vManage. If these are found in the list, vBond then checks the 256-bit random value to check that it has been correctly signed using the vEdge's public key, which is gained from the board ID certificate. If this is correctly signed, then vBond uses the root CA chain to validate the board ID certificate.

If the board ID certificate is valid, then the authentication of vEdge to vBond is also complete. vBond now sends two messages. The first message is sent to the vEdge, and this contains the IP addresses and the serial number of the vManage and vSmart controllers in the network. The second message is sent to the vSmart controllers and contains a message announcing a new vEdge router in the network and a request for the vSmart to set up a session with the new router.

With this done, vEdge terminates the DTLS tunnel to vBond and starts a DTLS tunnel to vManage.

vManage sends its trusted root CA-signed certificate to vEdge, which vEdge uses to check the organization name. If this matches, then vEdge uses its own CA certificate to check that the certificate is valid. If both pass, then vManage has authenticated to vEdge.

If this is starting to sound familiar, it is because it is. The process then follows with vManage sending a 256-bit random value challenge to vEdge, and vEdge sends the same details that it sent to vBond, to vManage. vManage then performs the same checks against this, checking the serial and chassis numbers, uses vEdge's board ID certificate to extract vEdge's public certificate, and checks the board ID certificate using the CA certificate. If these pass, then vEdge has authenticated to vEdge.

The next step is that vManage then sends the configuration file to the vEdge router (if one exists). Once received, this is activated, and vEdge starts advertising its prefixes to vSmart.

This kicks off another set of authentications, of vSmart to vEdge and vice versa, which once completed switches the temporary DTLS tunnel used for authentication to a permanent tunnel over which OMP sessions will run.

Alternative vEdge Deployments

As the vEdge device is the same as vBond, refer back to that chapter for the installation steps on KVM and VMWare.

vEdge in the Cloud

Viptela edge devices are also offered by several cloud providers, such as AWS and Azure, and in this step, we will look at how to deploy vEdge in AWS.

Start by navigating to the AWS Console (https://aws.amazon.com/console/), and then select the option for EC2. Select the Launch instance option in the main window (Figure 7-11).

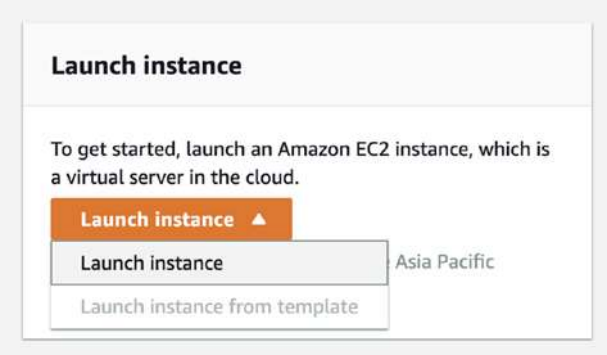

Figure 7-11. *Launching a vEdge EC2 instance*

In the net window, search for "vEdge," and select the AWS Marketplace option (Figure 7-12).

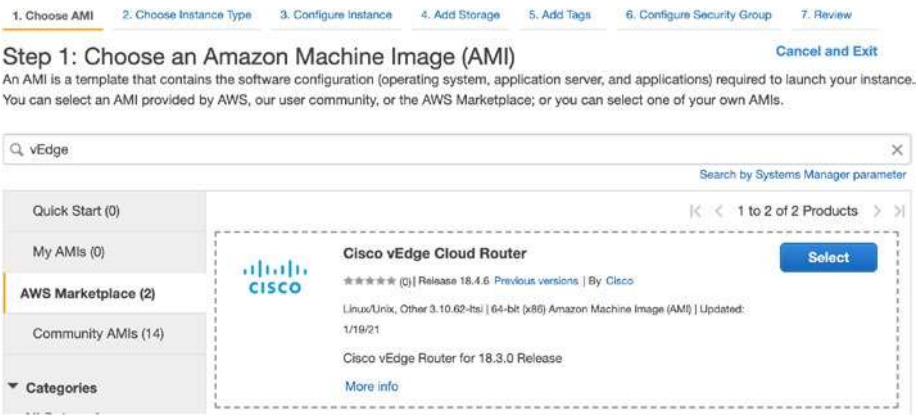

Figure 7-12. *The AWS marketplace*

Select the Cisco vEdge Cloud Router option, review the pricing details, and select Continue (Figure 7-13).

Figure 7-13. *Many different size options*

Choose your instance type, some of which are eligible for "free tier" (Figure 7-14).

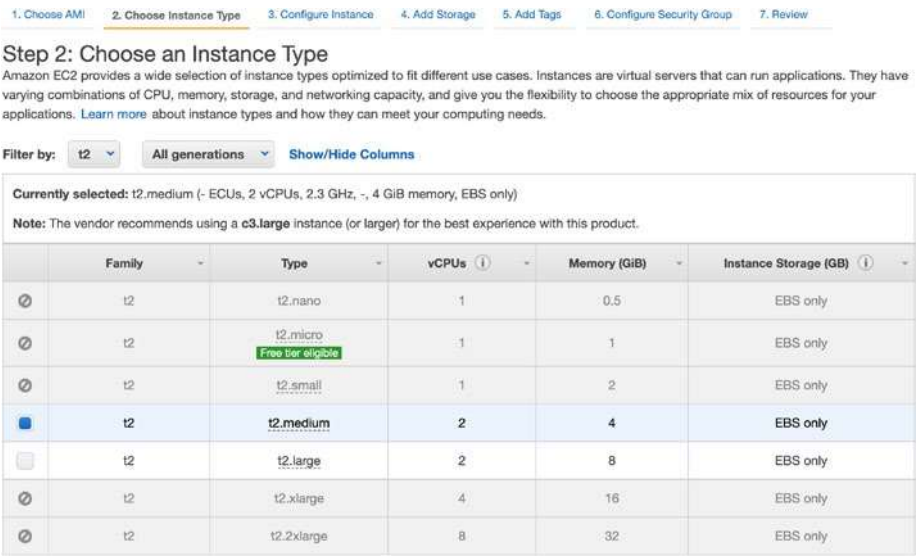

Figure 7-14. *Choosing the AWS instance type*

Review and launch (Figure 7-15).

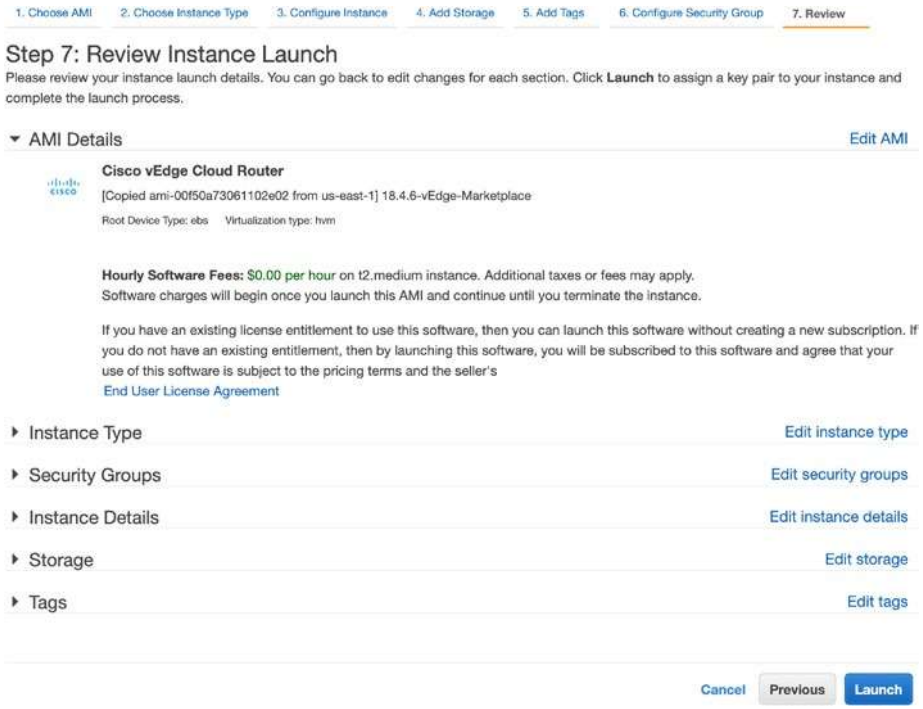

Figure 7-15. *Reviewing before launch*

Select an existing key pair, or create a new key pair (Figure 7-16). You do need to do this, even though some (older) documentation specifies to not use a key pair. If you are creating a new key pair, download it and keep it safe.

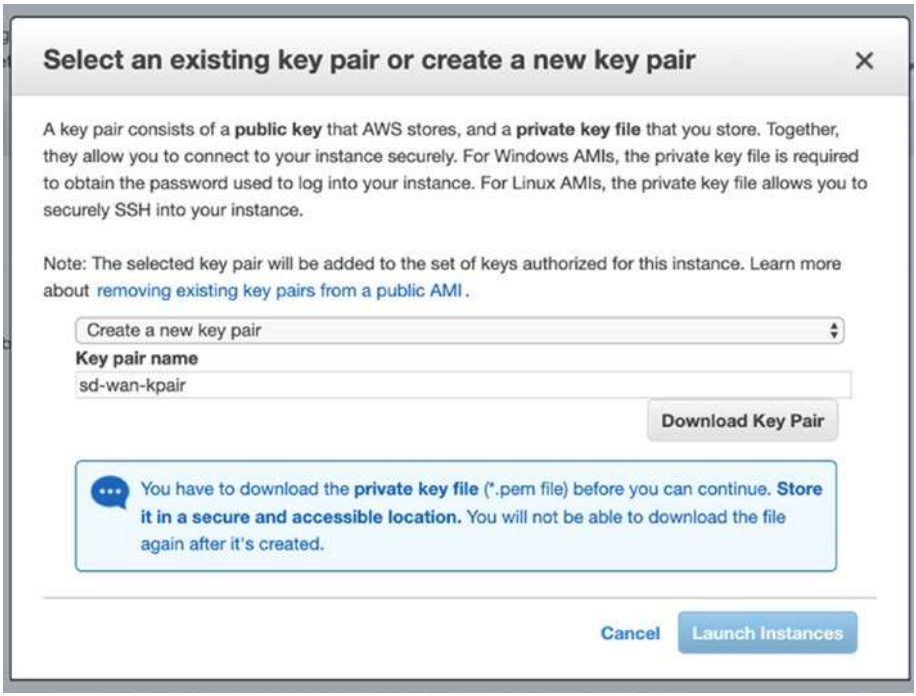

Figure 7-16. *The AWS key pair*

Click Launch Instances, which will start the build process (Figure 7-17).

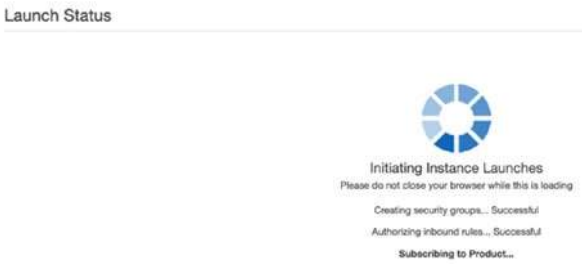

Figure 7-17. *Initiating the instance*

After a few moments, the instance will be ready for you to connect (Figure 7-18).

Figure 7-18. *The launched instance*

Once it is in a "running" state, click Connect (Figure 7-19).

Figure 7-19. *Connecting to the instance*

The next window will explain how to connect to your instance (Figure 7-20).

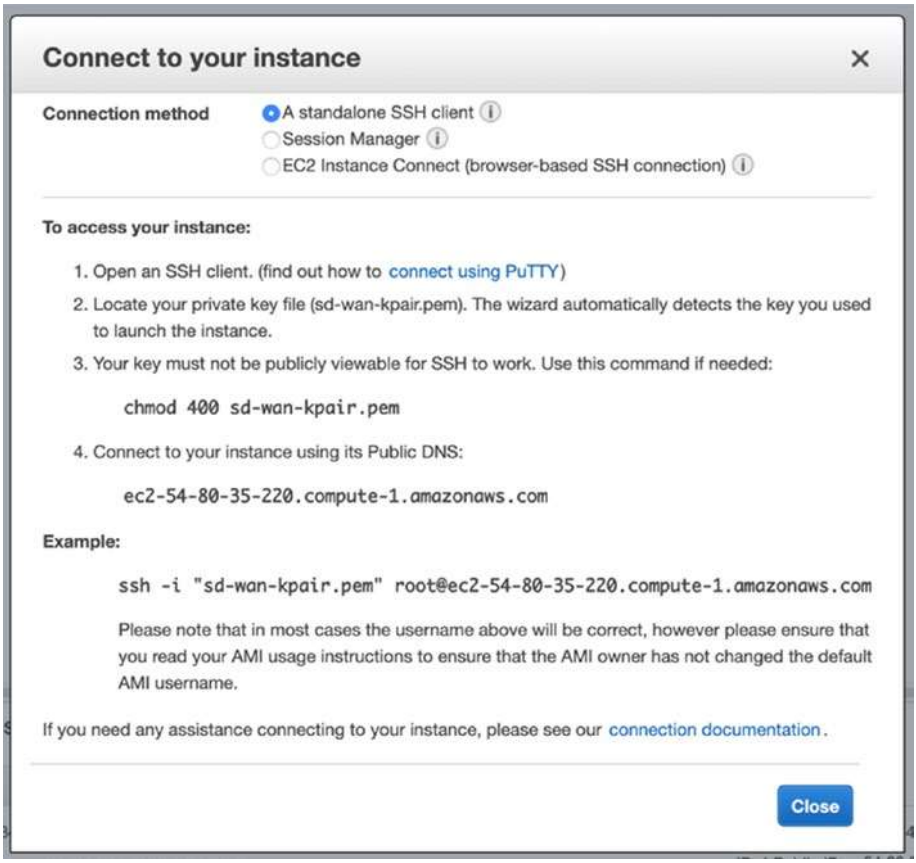

Figure 7-20. *Connection details*

Note, though, that we need to connect with the "admin" username, rather than "root," as shown:

```
iMac:Docs sfordham$ chmod 400 sd-wan-kpair.pem
iMac:Docs sfordham$ ssh -i "sd-wan-kpair.pem" root@ec2-54-
80-35-220.compute-1.amazonaws.com
The authenticity of host 'ec2-54-80-35-220.compute-1.amazonaws.
com (54.80.35.220)' can't be established.
ECDSA key fingerprint is SHA256:wzOA5ECpcg5v7eeVSMIOWmmBL2rqlF8
kmFXOXgh6tmc.
```

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-80-35-220.compute-1.
amazonaws.com,54.80.35.220' (ECDSA) to the list of known hosts.
viptela 18.3.0

Please login as the user "admin" rather than the user "root".

Connection to ec2-54-80-35-220.compute-1.amazonaws.com closed.
iMac:Docs sfordham$ ssh -i "sd-wan-kpair.pem" admin@ec2-54-
80-35-220.compute-1.amazonaws.com
viptela 18.3.0

Last login: Wed Oct 16 21:58:25 2019 from 54.185.130.54
Welcome to Viptela CLI
admin connected from 18.175.30.183 using ssh on vedge
vedge#
vedge# exit
Connection to ec2-54-80-35-220.compute-1.amazonaws.com closed.
iMac:Docs sfordham$
```

From there, the configuration is exactly as we have performed earlier in the chapter, and if your vBond is publicly accessible, the AWS vEdge will connect to it.

Preparing vEdge for ZTP

ZTP stands for zero-touch provisioning and allows us to configure vEdge devices to join the overlay just by powering them up! OK, so the process isn't *quite* that simple; we do need to make sure we have things set up already:

1. The site the new router is on must have access to `vtp.viptela.com`, and this must be reachable via public DNS servers (Cisco recommends using Google's DNS servers 8.8.8.8 and 8.8.4.4).

2. The correct ZTP interface must be connected. For vEdge 1000, this is ge0/0; for vEdge 2000, the interface to use is g2/0; and for the 100 series routers, it is ge0/4.

The process of zero-touch provisioning is as follows:

3. The router boots up.
4. The router tries to get an IP address from DHCP. With vEdge routers, if there is no DHCP server, then the router initiates an automatic IP address detection, by examining ARP packets on the network and using a free IP address in the range to assign to the ZTP interface. For the IOS XE routers, if there is no DHCP server available, then the ZTP process stops.[2]
5. The router sends a DNS request for `ztp.viptela.com`.
6. The vEdge connects to the ZTP server, which then verifies the router and sends it the IP address of the vBond orchestrator.
7. The vEdge router sends the chassis ID and serial number to the vBond orchestrator. Once the vBond has verified the chassis ID and serial number, it sends the vEdge router the IP address for the vManage NMS.
8. The router connects to and is verified by the vManage NMS, which then sends it back its system IP.

[2] www.cisco.com/c/en/us/td/docs/routers/sdwan/configuration/sdwan-xe-gs-book/cisco-sd-wan-overlay-network-bringup.html

9. Now that the new vEdge router has a system IP, it reconnects to the vBond orchestrator using its new system IP.
10. The vEdge router also re-establishes a connection to the vManage NMS, and, if necessary and enforced, the NMS pushes a software image to the router, whereupon the router will install this and reboot.
11. After the reboot, the router re-establishes the connection to the vBond orchestrator and the vManage NMS. The vManage then pushes the full configuration to the router.
12. Now that the router has the full configuration, it joins the overlay network.

For this process to work, the vManage NMS requires device configuration templates for the ZTP routers; otherwise, the process will fail. While we cannot, in our lab environment, test out ZTP fully, in the next chapter, we are going to start creating and applying some templates and correct some of the "misconfigurations" created in this chapter.

Summary

In this chapter, we set up our three edge devices and started to advertise some routes into our network. We also looked at cloud deployment and zero-touch provisioning.

CHAPTER 8

Templates

In this chapter, we are going to look at, create, and apply templates to our devices. Using templates, the edge device configurations will, for the most part, be the same, leaving us just to fill in the parts that are unique (such as IP addresses).

We are going to start this by putting things right which once went wrong. Just like Doctor Samuel Beckett.

We start with vEdge02 which, according to the topology, should be in site 200, so let's put this into the correct site:

```
vEdge02# config
Entering configuration mode terminal
vEdge02(config)# system
vEdge02(config-system)# site-id 200
vEdge02(config-system)# commit and-quit
Commit complete.
vEdge02#
```

We can see that the overlay network updates quickly to reflect the change:

```
vBond01# show orchestrator connections
```

```
          PEER    PEER     PEER           SITE  DOMAIN
INSTANCE  TYPE    PROTOCOL SYSTEM IP      ID    ID IP
-------------------------------------------------------
0         vedge   dtls     100.100.1.5    100   1
0         vedge   dtls     70.100.100.1   100   1
0         vedge   dtls     60.100.100.1   200   1
0         vsmart  dtls     100.100.1.4    100   1
0         vsmart  dtls     100.100.1.4    100   1
0         vmanage dtls     100.100.1.2    100   0
0         vmanage dtls     100.100.1.2    100   0
0         vmanage dtls     100.100.1.2    100   0
0         vmanage dtls     100.100.1.2    100   0

vBond01#
```

Now let's try the same with CSR-1, which should be in site 300:

```
CSR-1#config-transaction

admin connected from 127.0.0.1 using console on CSR-1
CSR-1(config)# sdwan
CSR-1(config)# system
CSR-1(config-system)# site-id 300
CSR-1(config-system)# commit
Aborted: 'system is-vmanaged': This device is being managed by
the vManage. Configuration through the CLI is not allowed.
CSR-1(config-system)#
```

Now, this is why we made the "misconfigurations" in the last chapter, so we can see what kind of issues we might run into. With this in mind, we are going to look at templates and the caveats that come with them.

Templates work well, but they can also completely hose your configurations if implemented incorrectly.

The example here is that I created a basic template that included a new user. I assigned it to the CSR-1, which was then downloaded and applied to the router. So far, so good, I thought.

I then lost connectivity between the new router and vSmart, and I could see this on the router console:

```
OMPD: vSmart peer 100.100.1.4 state changed to Init
OMPD: vSmart peer 100.100.1.4 state changed to Handshake
OMPD: vSmart peer 100.100.1.4 state changed to Up
OMPD: Number of vSmarts connected : 1
Line protocol on Interface Tunnel1, changed state to down
Line protocol on Interface Tunnel1, changed state to up
Configured from NETCONF/RESTCONF by vmanage-admin, transaction-id 591
Line protocol on Interface NVI0, changed state to up
OMPD: vSmart peer 100.100.1.4 state changed to Init
OMPD: Number of vSmarts connected : 0
OMPD: Operational state changed to DOWN
OMPD: Operational state changed to UP
OMPD: vSmart peer 100.100.1.4 state changed to Init
OMPD: vSmart peer 100.100.1.4 state changed to Handshake
OMPD: vSmart peer 100.100.1.4 state changed to Up
OMPD: Number of vSmarts connected : 1
Line protocol on Interface Tunnel1, changed state to down
Line protocol on Interface Tunnel1, changed state to up
Configured from NETCONF/RESTCONF by system, transaction-id 627
OMPD: vSmart peer 100.100.1.4 state changed to Init
OMPD: Number of vSmarts connected : 0
OMPD: Using empty policy from peer 100.100.1.4
```

I have truncated the output, but you should be able to see that initially we connect to vSmart, we receive a new configuration, courtesy of NETCONF, and then lose our vSmart connection.

So, what went wrong?

Programmatically, nothing went wrong. The overlay did exactly as it was told. It applied the template as it was instructed. From an operator standpoint, everything went wrong. The template had several defaults left in it, one of which was this:

```
interface GigabitEthernet1
 no shutdown
 arp timeout 1200
 ip address dhcp client-id GigabitEthernet1
 ip redirects
 ip dhcp client default-router distance 1
 ip mtu    1500
 mtu 1500
 negotiation auto
exit
```

Instead of the IP address I set in the previous chapter, the interface was now set for DHCP, and the admin password had changed (again, according to the template default).

Taking into account the ease with which you can turn a working network into a non-working network through applying badly configured templates, we are going to tread very lightly!

Creating Templates

The basic process of templating is that we create a template and attach a device (or devices) to that template and then vManage pushes it out to the respective endpoints.

We can create two types of templates: device and feature. Device templates are assigned to devices, and feature templates target specific areas of the configuration, such as AAA, but are also for specific devices. Don't worry, this will make more sense as we move forward.

Device templates are essentially an amalgamation of different feature templates. We do not cover CLI templates here, but these allow us to create the templates by typing (or pasting) the configuration directly in the browser.

We start by navigating to *Configuration* ➤ *Templates* (Figure 8-1).

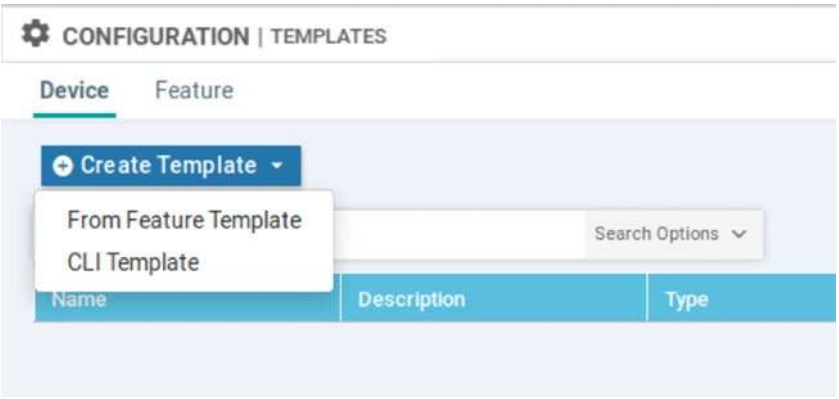

Figure 8-1. *Creating our first template*

The choices here are Device or Feature. Let's start by creating some feature templates. Select the Feature option at the top of the page, and then click "Add Template."

We won't see the available options until we select a device model, or models (Figure 8-2).

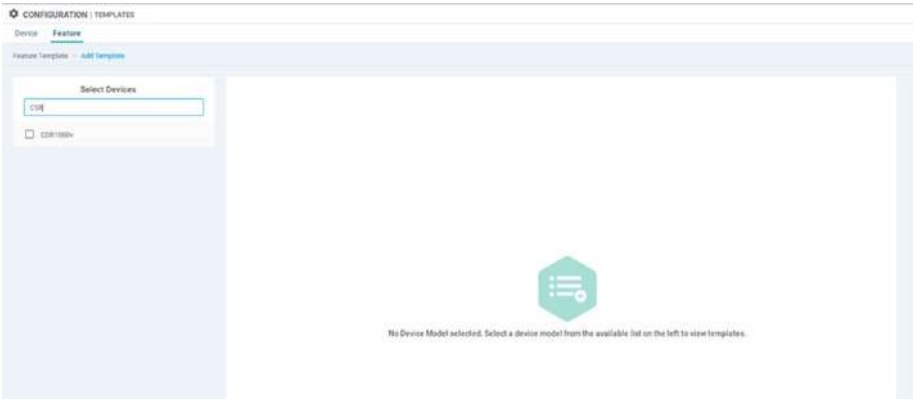

Figure 8-2. *Select a device*

We can select multiple models as some templates will be more generic than others, and having the same code-base or command structure means that one template can span multiple different devices. Our first feature template will be for local users. However, as we will see, sometimes it's not easy to create one template for everything.

Start by searching for "cloud" and selecting the vEdge Cloud device (Figure 8-3).

Figure 8-3. *vEdge Cloud templates*

You will see that we now have some options, including AAA, which is what we need. Now search for CSR, and select the CSR1000v device (Figure 8-4).

Figure 8-4. *CSR1000v templates*

Notice that we have now lost AAA, as well as Archive. This is because we cannot use the same code for creating the same user. On the vEdge routers, the command syntax would be

```
vEdge01# config
Entering configuration mode terminal
vEdge01(config)# system
vEdge01(config-system)# aaa
vEdge01(config-aaa)# user test
vEdge01(config-user-test)# exit
vEdge01(config-aaa)#
```

Or simply

```
vEdge01(config)# system aaa user test
vEdge01(config-user-test)#
```

But for the CSR router, we do not need to use such a long command to achieve the same result:

```
CSR-1#config-transaction
CSR-1(config)# user test
CSR-1(config-user-test)# exit
CSR-1(config)#
```

Therefore, we will have to create two templates, one for each of our router types. Deselect the CSR router from the left-hand side, so that we are left with just the vEdge Cloud ticked. Select AAA from the options under Basic Information when it reappears. Call the new template "v-AAA" (the "v" is for vEdge), and give it a description (Figure 8-5).

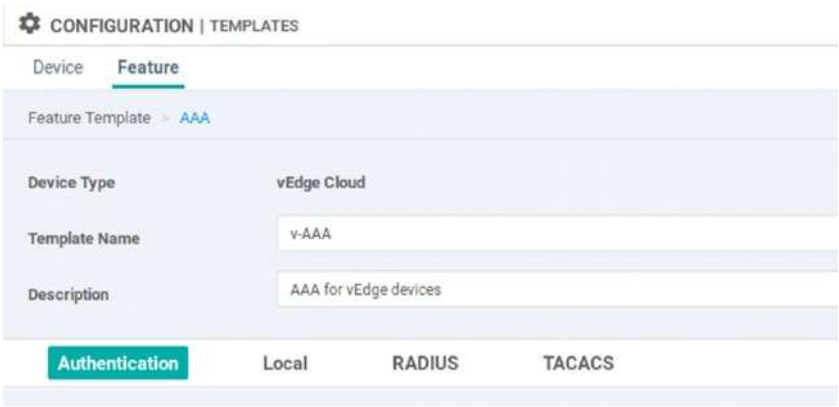

Figure 8-5. *v-AAA template*

By default, the "admin" user has a password of "admin" (Figure 8-6). If we apply the template as it currently stands, then it will overwrite the password set when the router was set up, so let's start by changing the default.

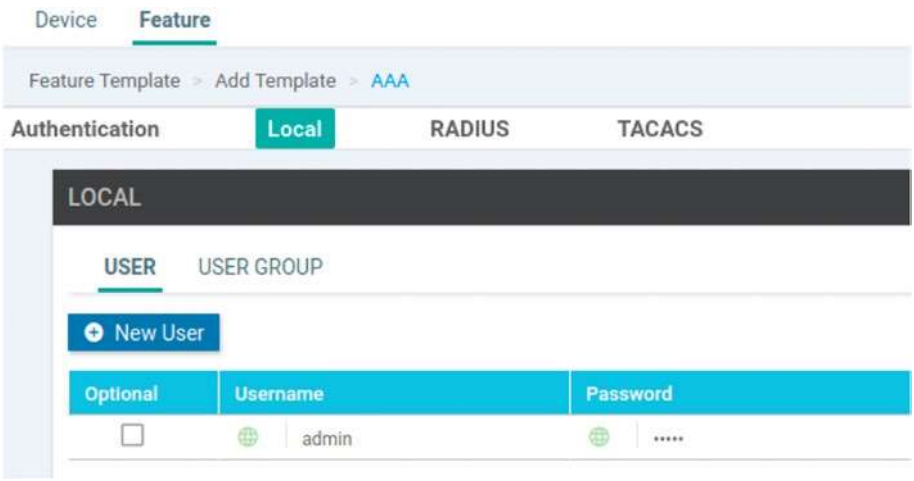

Figure 8-6. *The default admin user*

Click the action icon at the right-hand side to bring up the user properties window. Change the password to something you prefer, and click "Save Changes" (Figure 8-7).

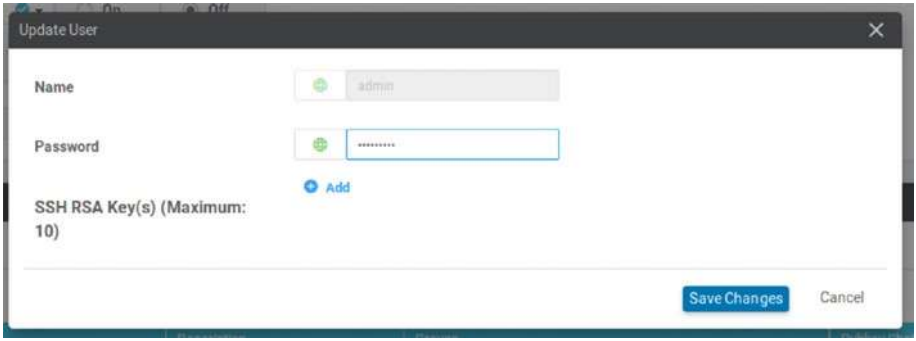

Figure 8-7. *Editing the admin user*

Now click "New User," and create a user called "net-ops" making them a member of the Operators group (Figure 8-8).

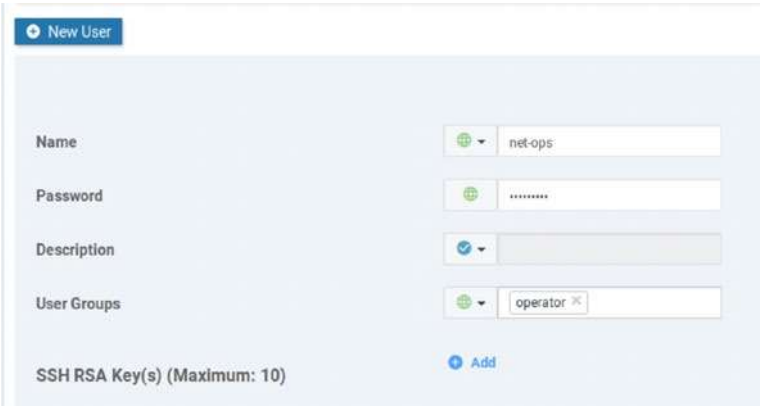

Figure 8-8. *Creating a new user*