LEARNING MADE EASY

VMware® Special Edition

Network Automation

for dummies®
A Wiley Brand

Identify why automation is important

Examine how automation can help you

Find out how to automate

Brought to you by

vmware®

Karl Fultz
Madhukar Krishnarao
Susan Wu

Network Automation

VMware® Special Edition

by Karl Fultz, Madhukar Krishnarao, and Susan Wu

for dummies®
A Wiley Brand

Network Automation For Dummies®, VMware® Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-69977-4 (pbk); ISBN 978-1-119-69984-2 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Project and Development Editor: Chad R. Sievers
Associate Publisher: Katie Mohr
Editorial Manager: Rev Mengle
Business Development Representative: Karen Hattan
Production Editor: Siddique Shaik

Contents at a Glance

Table of Contents

Introduction

Automation is the current state of networking.

General purpose hardware, monitoring tools, and software-defined networking (SDN) are making it possible today.

You need a growth mindset and a team orientation to automate your network because IT tends to work in silos. Organizations have been shown to operate more efficiently when networking and DevOps work together on the same set of objectives. Network automation is a fundamental piece of the process.

About This Book

Network Automation For Dummies, VMware Special Edition, helps you understand these important network automation principles:

- **Make your network visible.** You can't manage what you can't see. The first step in automating a task is to understand the current functionality and processes. Get an inventory list of all network processes you're currently doing.
- **Use software-defined networking infrastructure.** Software-defined infrastructure (SDN) is a core principle that makes automation easier. SDN allows you to control your network through code. By enabling SDN, you can create replicable situations across hardware. Replicability is a core element to network automation.
- **Automate manual tasks.** Manual tasks make automation difficult. Turning network and security appliances into software and applying a common network virtualization model across the entire network makes it easier to automate manual tasks. Automating manual configurations tasks using RESTful APIs can help eliminate one-off scripts.
- **Keep the network simple.** Converge network operations with common tooling. Typical change requests often take several team members from networking and security silos working with disparate systems in order to complete.

For network automation to be successful, it's important to take an end-to-end approach and use programmable interfaces to automate the whole process, not just repetitive tasks like CLI commands on a few network devices.

The approach should be to build tools that people will use. Instrumentation is gold.

- **Build exception-based notification.** Notify the right people only when there's a problem. With a network monitoring system, you can monitor remote systems from a central location and send notifications automatically by simple network management protocol (SNMP), email, text, and voice. Screen messages and send only the most important ones to the central hosts.

When automation becomes part of the job description, people are empowered to implement improvements into day-to-day operations. Even though human error is one of the leading causes of network failures, the same people can build the automation rules that can save on operational expenses and can help move the talented staff to higher value projects.

Network automation may appear daunting at first, but it can be beneficial to any organization. Automation happens with small steps. Start by automating your mundane tasks and show early results, and then tackle the end-to-end processes to make yourself more valuable to your organization.

This handy guide consists of five chapters to help you navigate network automation in your organization. The chapters cover the following:

- The seven reasons for network automation (Chapter 1)
- A look at the network automation landscape, including NSX RESTful APIs, Policy API Model, and authentication (Chapter 2)
- The ins and outs of automating network services with vRealize Automation, infrastructure as code with blueprints (Chapter 3)
- A glance at SDKs and opensource automation tools (Chapter 4)
- Ten ways to start with network automation (Chapter 5)

Foolish Assumptions

When writing this book, we make the following assumptions about you, the reader:

- You know the basics about networking.
- You're familiar with network virtualization.
- You understand at a high level the infrastructure-as-code concept.
- You're aware of many of the DevOps processes and tools.

Icons Used in This Book

This book uses the following icons to call your attention to information you may find helpful in particular ways.

REMEMBER

The icon marks noteworthy information that you can refer to again and again.

TIP

This one helps you take action.

WARNING

Paragraphs marked with this icon call attention to common pitfalls that you may encounter.

Where to Go from Here

"Automate everything. Automate the automation."

ShaColby Jackson, Director Network Operations, BlueJeans

You don't need to read this guide from cover to cover. Just jump to a specific chapter that interests you and start reading. You can go back later and read any chapters you skipped to make sure you don't miss any vital information, though. We include code samples throughout the book to aid your understanding and for you to try out in your own environment.

For the latest news and information, visit `www.vmware.com/solutions/network-automation`.

IN THIS CHAPTER

- Making changes consistently
- Providing faster response time
- Helping network engineers to better solve problems

Chapter 1
Examining the Top Seven Reasons Why Network Automation Is Important

Network automation, like most forms of automation, is a means to doing things faster. Even though improving the speed of service delivery is a clear benefit, the decision to automate is usually more complex, often driven by business goals and the need for a more reliable, scalable network.

As more and more use cases emerge for API-driven architecture, network automation is the prime way to take advantage of the programmatic interface exposed by modern network solutions that offer an application programming interface (API). The advantage of API-driven management is that APIs deliver structured data rather than raw text (for instance, the output of a log) that network engineers can use to streamline day-to-day operations of managing networks, resolving network issues, and performing deep network analysis.

This chapter identifies the main reasons for implementing network automation to help you understand how your organization can benefit by using it.

Automating the Repetitive Tasks and Replicating with Templates

Much of a network engineer's job involves the command line interface (CLI), and much of that work involves syntax-specific keywords and phrases that are often repeated several times, depending on the change. Manually performing configuration tasks is inefficient and prone to errors. In fact, some industry reports suggest human error causes at least 40 percent of network failures (some estimates are as high as 80 percent).

REMEMBER

Network automation provides two important benefits to address these errors:

- **Consistency:** You're able to predictably and repeatably make changes to product networks and achieve the desired result. Network automation allows for the creation of a standardized base template for the organization and the network engineers and consumers (Help Desk, IT engineers, NOC) of the network to dynamically fill in some values as needed.
- **Increased productivity:** With repetitive tasks removed, you can focus on strategic activities like identifying new opportunities or driving new business improvements.

Providing Faster Service

With the advent of cloud and DevOps, you can deploy new applications almost instantaneously. The faster the applications are deployed, the more questions are raised as to why configuring and provisioning network services like VLANs, routes, firewall policies, and load balancing policies for the applications takes so long. The reason is because some organizations are still updating the configurations of routers and switches or changing firewalls manually.

REMEMBER

By automating manual updates and changes, you as a network engineer can provide faster service delivery, optimize network performance, and accelerate the rollout of new services and applications.

Standardizing Processes with the Use of Templates

Network automation allows you to declare which parts of the templates remain static and which parts should be dynamic, standardizing your processes.

For example, programming languages like Python have templating engines like Django and Jinga, and programming languages like Go and Ruby have in-built template systems. Instead of entering CLI commands, these templating engines make your job easier.

REMEMBER

By storing the syntax (which is static) and data (which is dynamic) in separate files, you can reuse the templates many times over because the data stored in its own YAML file can be imported into the existing template.

The following example shows how templating standardizes the process by using a loop to create configurations for ten switchports.

```
{% for n in range (10) %}
Interface GigabitEthernet0/{{ n+1}}
 Description {{ interface.description }}
 Switchport access vlan {{ interface.vlan }}
 Switchport mode access
{% endfor %}
```

In this template, you're calling the range() function to give you the list of integers to iterate over, and for each iteration, you print the result of "n+1" because the range starts at 0, and normally switchports start at 1.

You've just created an identical configuration for all ten switchports with GigabitEthernet0/1-10 in access mode using a single template instead of doing this task ten times in the CLI.

Making Changes Easier

Many network management processes aren't performed regularly because they're manual, time consuming, and resource intensive. Network automation makes it possible to perform these processes more frequently, reducing the risk of network failure and downtime. Also, when the network is automated, configurations are consistently applied across the infrastructure with less effort, simplifying network management for Day 2 operations.

Building a More Stable, Reliable Network

Infrastructure as code is managing your IT infrastructure using configuration files and applying the same source control techniques used in software development. In other words, it means maintaining the state and configuration of your infrastructure with the same processes developers use to manage source code.

Much like application code, configuration files for compute, storage, networking, and other resources can be versioned, peer reviewed, approved, merged, staged, and tested before deploying into production. Gone are the days when the human is in the direct control path of the network, making changes in production networks through the CLI. Humans are still involved but are in control of improving the automation process instead of being bogged down by doing the discrete tasks.

Meanwhile, *network as code* is applying the same infrastructure-as-code principles to the network domain from the data center to the edge to multicloud.

The three core principles in network as code are as follows:

- Store network configurations in source control, which is the single source of truth.
- Deploy configurations with programmatic APIs.
- Use a CI/CD pipeline to automate the build process.

The advantages of adopting network as code are many:

- You have the most current and comprehensive view of the network configurations for your devices at any given time.
- You can verify only the desired changes have taken place through the version control, which is the system of record.
- You can keep configuration templates in source control, making them easier for reuse.
- You can use source control with network documentation.

An added benefit is accountability. When an individual makes a change, network as code tracks the change, ending any finger-pointing and providing an audit trail for regulatory compliance.

Allowing You to Monitor Performance

One of the core functions in network management is performance monitoring. Although this function appears to be fundamental, it can be a challenge for networks comprised of hundreds of switches and routers without automation due to vendor-specific implementations and changes in standards. Network automation supports monitoring tools that alert you to performance issues, high resource utilization levels, and errors on the network. It also allows you to perform and report upon service-level agreement tests with greater speed and accuracy.

Analyzing Issues and Solving Them Faster

Even though most network management systems collect the network data, more than likely you're challenged in identifying and troubleshooting problems without knowing the network topology. Keeping the network topology drawings updated is a tedious and often neglected task in the era of having to do more with less.

Complex network analysis involves multiple sources of data, such as from configuration files, events, and operational data. The combination of network automation and network analytics can provide deeper insights into performance, utilization, security, and resource allocation and help you automate analysis tasks and resolve issues more quickly than manual techniques.

IN THIS CHAPTER

- Recognizing what the automation landscape is
- Getting to know the NSX RESTful APIs
- Comprehending NSX authentication

Chapter 2
Understanding the Ins and Outs of NSX Automation

Developers, app owners, network admins, and security engineers all expect a quick turnaround on resource request. However, limited management tools, lack of governance, and cumbersome processes often create a barrier for quick turnarounds. As a result, as an organization, the best way to address these issues is twofold:

- Target the overarching process to deliver a complete infrastructure plus application resources.
- Abstract away the complexity of network and software operations through network virtualization.

Although the automation of compute and storage has become mainstream through vSphere, automation of NSX provides an easy way to enable network virtualization.

This chapter examines NSX automation in greater detail and explains the different ways that network automation can work for your organization.

Looking at the Automation Landscape

NSX provides a consistent layer for interaction across various systems like automation tools and programming languages. All interactions with NSX can be broadly classified as one among the following, as Figure 2-1 shows:

- Content management systems
- Programming languages/SDKs
- Configuration management tools

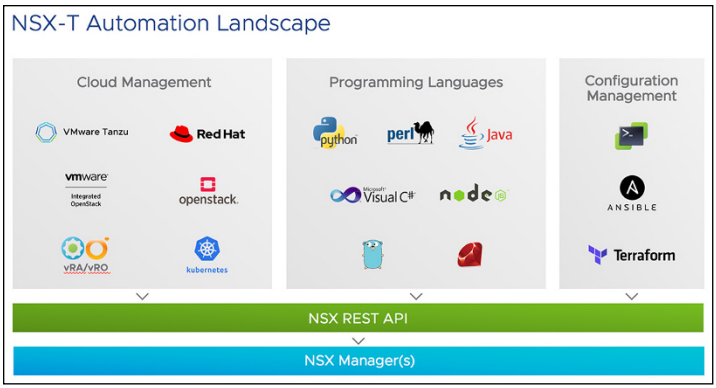


FIGURE 2-1: The landscape overview.

Although NSX provides a whole set of options, all interaction with NSX happens through the RESTful (representational state transfer) APIs (also referred in this book as REST APIs). The application programming interfaces (APIs) provide a single consistent platform and entry point for all external requests. The following sections examine each of the three categories in detail and discuss different options that are available.

Content management systems

Content management systems can be classified in two categories:

- Orchestration systems like vRealize Automation or vCloud Director

» KVM hypervisor management systems like VMware Integrated OpenStack, OpenStack, DevStack, or K8s platforms like Pivotal Container Service and OpenShift

These systems can deal with compute, network, and storage virtualizations. NSX provides the networking aspect and integrates cleanly with these systems. You can continue to use your favorite system and still use NSX underneath.

Programming languages/SDKs

Programming languages/SDKs are one of the most popular tools when looking at automation. The SDKs provide the library that exposes functions that can interact with all the various NSX objects. NSX provides ready-to-use SDKs in Python and Java available for download. However, the support for SDKs doesn't stop with these two programming languages. NSX APIs are based on OpenAPI spec, which means you can generate your own SDK in your own favorite programming language.

Here's how you can do it:

1. **Download the NSX API spec as shown:**

```
GET /api/v1/spec/openapi/nsx_policy_api.yml and
GET /api/v1/spec/openapi/nsx_api.yml
```

2. **Run it through swagger-codegen as shown:**

```
Swagger-codegen generate -I nsx_policy_api.yml -l php
```

or

```
Swagger-codegen generate -i nsx_policy_api.yml -l ruby
```

Configuration management tools

You can also use popular configuration management tools like Ansible or Terraform to manage NSX. Going with the spirit of opensource, the Ansible NSX modules and NSX Terraform provider are available for free, are opensource, and are fully supported. You can create and save your entire network configuration in a playbook or a manifest file. You can then revision control these configurations, making a true infrastructure-as-code solution. Chapter 3 looks more into some of these options.

Explaining NSX REST APIs

NSX REST APIs provide the single source of entry into managing your NSX environment. Via the APIs you can deploy additional NSX nodes, prep your transport nodes, and create the desired network topology.

Table 2-1 introduces and defines the standard verbs that NSX REST APIs support:

TABLE 2-1 Standard Verbs

Supported Verbs	Definition
GET	Retrieve data about a single NSX object or multiple objects.
POST	Create an NSX object.
PUT	Modify properties of an existing object.
PATCH	Edit if exists. Create instead.
DELETE	Remove an NSX object.

Here is what a typical API REST resource URL looks like:

```
https://nsxmgr-01.corp.local/policy/api/v1/infra/
  segments
```

The resources can be classified as such:

- Protocol to connect to the API
- FQDN or IP address of NSX manager
- Base API path
- Resource

The following sections examine the REST APIs in greater detail.

OpenAPI

The REST APIs in NSX manager is based on OpenAPI spec. OpenAPI is a language based on YAML syntax and is used to describe an

API in a human-friendly way. Even though the syntax is human friendly, it can still be parsed programmatically to build documentation and dynamic clients. In fact, NSX APIs have been following the OpenAPI specifications since version NSX-T 1.1.

That's great, but how does that help you? Well, because the REST APIs are based on the OpenAPI spec, not only can the VWware/NSX team build the API Guide automatically, ensuring that it's always in sync with the actual API call and reflects the changes, but it also means you can import the API spec into any standard REST client. This makes working with APIs quite easy, and you don't have to remember each and every API.

You also can view the NSX API Guide right in the product. After you're logged in, just click on the (?) on the top right of the product user interface (just left of the username). It opens up a menu that has a link API documentation.

API rate limiting

NSX REST API does have some resource requirements. To counter any potential issues, the NSX manager has checks in place so that it doesn't get overwhelmed in just servicing the API requests that come from a single source or from multiple clients. In order that all clients' requests get serviced, there has to be a fair distribution of resources. To address this, NSX APIs enforce rate limits. Table 2-2 shows the different limits being enforced:

TABLE 2-2 NSX APIs Enforcement Limits

Type	Limit	Error When Limit Is Reached
Client API Rate limit	100 requests/sec	HTTP 429
Client API concurrency limit	40 connections	HTTP 429
Global API concurrency limit	199 requests	HTTP 503

Note: Each NSX manager has defined limits, and you can change them. To do so, use nsxcli, the NSX command line interface. The question though is should you change it. That depends.

WARNING

Be careful changing the rate limits because doing so can affect the performance of the NSX manager. We suggest you first consult with your trusted NSX support representative before making any changes.

NSX Policy APIs

NSX-T release 2.4 introduced a new object model to simplify and automate network and security configurations through outcome driven statements. The resulting new Policy APIs do the following:

- Reduce the number of configuration steps by allowing users to describe the desired end goal while letting the system figure out how best to achieve it.
- Create the entire intent in one go in an order-independent prescriptive manner. This handy guide provides a quick guide to understand the new Policy API model, covers the consumption, and talks about the hierarchical API it provides.

  Don't confuse the Policy API with a Security Policy (in a DFW context).
- Provide a simplified data model and allow for consumption using an intent-based approach. They use a declarative API model and can be used to define the entire intent in one API call. But what about object dependencies? You can define objects in any order, and the framework will figure out which objects have to be created first making the Policy APIs order independent.

  The data model guarantees that a parent object is created before its child object, making it order independent. The model also allows for a user-defined Object ID to be specified while creating an object. Each object can be referenced by providing the full path of the object hierarchy. The APIs operate on policy objects and are provided under the hierarchical API endpoint as such: `/policy/api/v1/infra/`


REMEMBER

Figure 2-2 highlights a high-level overview of the object hierarchy:

As Figure 2-2 shows, the specific object follows the tree structure. For example, referring to a rule would be through: `/infra/domain/security-policies/rules/`.

Note that it's not the API path but the path to the resource. The API path would be `/policy/api/v1/infra/domain//security-policies/rules/`.

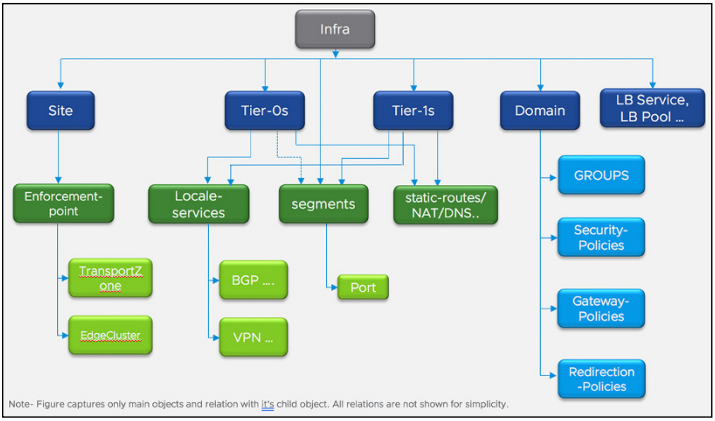


FIGURE 2-2: Object hierarchy.

With the introduction of Policy APIs, the new objects have a new naming scheme. Table 2-3 lists the new Policy constructs and their corresponding older constructs. The older constructs are still valid and are used when using the traditional (MP) APIs.

TABLE 2-3 The Existing and New Constructs

Existing Construct	New Policy Construct	Definition
Logical Switch	Segment	A network entity equivalent to Logical Switch.
T1 Logical Router	Tier-1 Gateway	Equivalent to the T1 Router and allows the topology to scale out. Multiple Tier-1 Gateways talk to the Tier-0 Gateway.
T0 Logical Router	Tier-0 Gateway	Equivalent to the T0 Logical Router and allows Tier-1 s to talk to the outside world.
NSGroups, IP Sets, MAC Sets	Group	Grouping construct to statically or dynamically group different objects, such as inventory entities like IPs, VMs, MACs, and so forth.
Firewall Section	Security Policy	A section to encompass various security policies. Each Security Policy has a set of Firewall Rules.
Firewall Rule	Rule	A structure to encompass various firewall policies.
Edge Firewall	Gateway Firewall	Tier-0/Tier-1 Edge Firewall capabilities for North-South connectivity.

Policy API Data model

Policy APIs are built as regular REST APIs, and they support the traditional GET/PUT/DELETE calls. Where applicable, the calls accept a JSON body formatted in a specific way (as defined by the schema in the API Guide) and return a code to indicate success or failure.

One of the changes that has a big impact is the ability to use user-defined IDs, which can be alpha numeric. They're also used as the display_name if one isn't specified, which allows for objects to be easily identified and searched for. It allows for the Object ID/Object Name to be from the URI. Consider a simple API call: `PATCH / policy/api/v1/tier-0s/MyTier0`

The Object ID is "MyTier0," and the display name and the fields need not be specified in the request body. Consider the following where the empty request body is valid to create a new Tier-0 Gateway (with default values for its members) because the resource type and Object ID/Object Name is inferred from the URI. With these two fields known, the rest of the fields use the default values:

```
PATCH /policy/api/v1/tier-0s/MyTier0

{}
```

Hierarchical API model

The API model also follows a parent-child hierarchical tree structure. Multiple objects can be defined as a nested parent-child tree, which helps define multiple objects using one single API call. Consider this API endpoint: `/policy/api/v1/infra/`.

It's treated as the root object, and you can do CRUD (Create/Read/Update/Delete) operations on this endpoint. Just by interacting with the endpoint, you can create or modify entire topologies.

For example, in the following API call, a Tier1 Gateway object and a Segment can be operated on using the singe PATCH API call. If the objects exist, they'll be modified to match the request body. If the objects don't exist, they'll be created.

```
{
 "resource_type": "Infra",
 "id": "infra",
```

```
  "children": [
   {
    "resource_type": "ChildTier1",
       "marked_for_delete" "False",
       "Tier1": {
          "resource_type": "MyTier1",
          "id": "Tier-1",
          "children": [
            {
              "resource_type": "ChildSegment",
              "marked_for_delete": "false",
              "Segment": {
                "resource_type": "Segment",
                "type": "DISCONNECTED",
                "connectivity_path": "/infra/
  tier-1s/MyTier1",
                "transport_zone_path": "/infra/
  sites/default/enforcement-points/default/
  transport-zones/664ba01c-815d-48ba-a7e0-
  8ff1d928db50",
                "id": "MySegment",
                "children": []
             },
            }
          ]
        }
      }
  ]
}
```

Marked_for_delete flag

The last important field while working with Policy APIs you need to know is the marked_for_delete flag. It means exactly what it says. If the flag is true, then the items are deleted when the

PATCH call is run. When it's set to false, objects are left as is. Remember these important points:

- You can have the marked_for_delete flag for both parent and child objects or for specific objects.
- The flag at the parent level overwrites the child flags.
- This flag only operates when you do the PATCH API and only at the `/policy/api/v1/infra` endpoint.
- You can use the flag to create a few objects and delete a few more in one PATCH call.

You can find more information on the Policy APIs and the working examples in the "Getting Started Guide" on VMware Communities site (`https://communities.vmware.com/docs/DOC-41182`).

The guide also includes working examples and is a great place to start if you're starting to look at NSX Automation using REST APIs.


TIP

You can do a GET `/policy/api/v1/infra?Filter=Type-` API Call to get the entire NSX config. You can also filter on different object types.

Chapter 3 focuses on the possibilities of automating NSX using vRealize Automation's native NSX integrations. There are scenarios where direct access to the NSX APIs is warranted. It's also possible to leverage vRealize Automation, in conjunction with vRealize Orchestrator and Action Based Extensibility, to directly access the NSX APIs as part of a deployment workflow and for day-2 actions.

Focusing on the Four Ways of Authentication

Authentication and authorization are important aspects when considering automation. The automation written has to cater to different types of users trying to authenticate themselves from different systems. NSX deals with this by providing four main ways of authentication:

- Basic authentication using username and password
- Session-based authentication
- Principal identity- (certificate-) based authentication
- vIDM (single sign-on–based) authentication

REMEMBER

While using the NSX user interface, basic authentication and vIDM-based authentication (the single sign-on solution for NSX) are the only types supported.

In the following sections we examine these four ways in greater detail.

Basic authentication

Basic authentication is one of the simplest authentication mechanisms to use. It expects the username and password to be sent in each API call for authentication. Here is an example:

```
curl –k –u admin:secretPw99 https://192.168.22.32/
  policy/api/v1/infra/segments
```

Session-based authentication

This way of authentication still uses a username and password, but a session cookie is first generated upon successful authentication with the username and password. All subsequent API calls send the session cookie instead of the username and password. Sometimes sending the session cookie back and forth with the server is ideal rather than sending the username and password. ***Remember:*** The session cookie is local to a server, so if you've received the session cookie from one NSX manager, you can't use it to authenticate yourself on another NSX manager (within the same cluster).

In this following example, the CURL command will authenticate to the server, will deposit the session cookie in the file cookies.txt, and will write all HTTP response headers to the file headers.txt. One of these headers is the X-XSRF-TOKEN header that you must provide in subsequent requests:

```
curl –k –c cookies.txt –D headers.txt –X POST –d
  'j_username=admin&j_password=secretPw99'
  https://192.168.22.32/api/session/create
```

You must use the session cookie for future API calls. Along with the cookie, the X-XSRF-TOKEN header must be provided in subsequent requests. This header is part of the headers.txt file that gets created when the session cookie is generated.

For example:

```
curl -k -b cookies.txt -H "`grep X-XSRF-TOKEN
  headers.txt`" https://192.168.22.32/policy/api/
  v1/infra/segments/
```

When the session expires, the manager will respond with a 403 Forbidden HTTP response, at which point you must obtain a new session cookie and X-XSRF-TOKEN.

Default session expiry is set to 1,800 sec (30 mins). You can configure it by setting "*connection_timeout*" via API: `PUT https://<nsx-mgr>/api/v1/cluster/api-service`

You can also delete a valid session using the `/api/session/destroy` API:

```
curl -k -b cookies.txt -H "`grep X-XSRF-TOKEN
  headers.txt`" https192.168.22.32/api/session/
  destroy
```

Principal identity- (certificate-) based authentication

In this method, an SSL certificate, not a username or password, does the authentication. A certificate is associated with a user and is used for authentication and authorization. All calls made with the certificate identify the specific user. As an added benefit, it also provides a means of having object ownership. Objects created using this principal identity (PI) certificate are protected and only that user can edit or modify it. Other users can't overwrite it.

Another use of having the PI, perhaps the most important from an automation context, is to identify objects created through an external automation tool and to protect that tool from accidental edits. If administrators started editing objects created or owned by OpenStack or if one automation script accidentally deleted an object created by another automation script, it would be chaotic.

To prevent such accidental edits, PIs can be used to identify and protect objects. You can use one PI user with OpenStack and one PI user with K8 systems. Doing so will keep objects created by OpenStack and K8s protected from accidental edits.

Here's how it works. First, associate a certificate with a user identity. Then the certificate key is used while creating an object, which binds the object to the PI user and protects it. Edits to the object are possible only when the same certificate key is passed while calling the API call.

The administrator isn't able to modify the objects created by a specific PI, which is the whole advantage of using PI — object ownership and protection. But what if you really have to modify? Perhaps the PI user no longer is applicable or the tool that created it no longer exists. If so, NSX allows overwriting the protection by passing the *X-Allow-Overwrite* header while making the API call. Here's an example of how you can do it. In the example, the admin uses the overwrite flag to modify the T0 object owned by the PI:

```
curl --noproxy '*' -k -u admin -X PATCH
    https://192.168.22.32/policy/api/v1/infra/
  tier-0s/pi-SA-1-T0
    -H "Content-Type Application/json"
    -d '{"transit_subnets": ["10.1.1.0/24"], "ha_
  mode": "ACTIVE_STANDBY", "display_name":
  "pi-SA-1-T0-modifiedBy2" }'
    -H "X-Allow-Overwrite: true"
```

Note only through the APIs can you create and modify objects through a PI.

vIDM/Single sign-on–based authentication

As organizations grow larger, they typically execute user management with systems such as Active Directory or LDAP. NSX has the ability to integrate with such systems through VMware Identity Manager (vIDM). You register the instance of vIDM with NSX and you can authenticate using the `user@domain` account. NSX passes the authentication information to vIDM and upon

successful authentication by vIDM, NSX Manager receives the *OAuth2AccessToken*.

NSX manager then issues a JWT Token (JSESSIONID) as part of the REST API response to the user. This Token expiration is set to 15 minutes, which is configurable via vIDM.

The Token is then used for further authentication until it expires. The NSX manager locally does the Token validation.

Nearing the expiration of the 0Auth2AccessToken, NSX manager presents a Refresh Token calls a vIDM REST API to obtain a new OAuth2AccessToken, as Figure 2-3 shows.

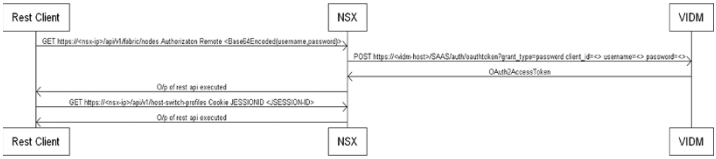


FIGURE 2-3: Interaction between NSX Manager and vIDM.

From an API or automation perspective, all the preceding interaction is hidden from the user. All the user has to do is pass the -H Authorization: Remote header!

```
curl -k -H "Authorization: Remote BASE64(user@
  domain:password)" https://nsx-mg/api/v1/
  logical-ports
```

IN THIS CHAPTER

- Explaining vRealize Automation – Cloud Assembly
- Using infrastructure-as-code blueprints
- Automating network and security services
- Utilizing IPAM solutions
- Providing a catalog for user requests

Chapter 3
Automating Network Services with vRealize Automation

This chapter covers how vRealize Automation and NSX work together to automate the creation and consumption of network and security resources. You find out how the network objects are created and managed by vRealize Automation during a deployment. We explain the benefits to using vRealize Automation and then examine the specific functionality areas that cover the integration between Cloud Assembly and NSX.

Understanding Cloud Assembly and Its Many Benefits

vRealize Automation and vRealize Automation Cloud include the containerized service Cloud Assembly. Cloud administrators will work primarily within *Cloud Assembly* to build out the ability to create provider and on-demand networks and use existing

software-defined networks through integrations with VMware NSX. Cloud Assembly also allows you to leverage the network resources provided by public cloud providers. Here are some of the advantages to using vRealize Automation – Cloud Assembly.

Governance

One of the many advantages of Cloud Assembly is the ability to provide organizations with control over resources under management. This control is the difference between a configuration management tool and hybrid-cloud management platform. Organizations and projects provide multi-tenancy, a boundary that controls access to policies, content, provisioned resources, and capacity (see Figure 3-1). Administrators add users to projects in order to control access to each item. Users are only able to request and consume network services that are associated with the project they have access to.

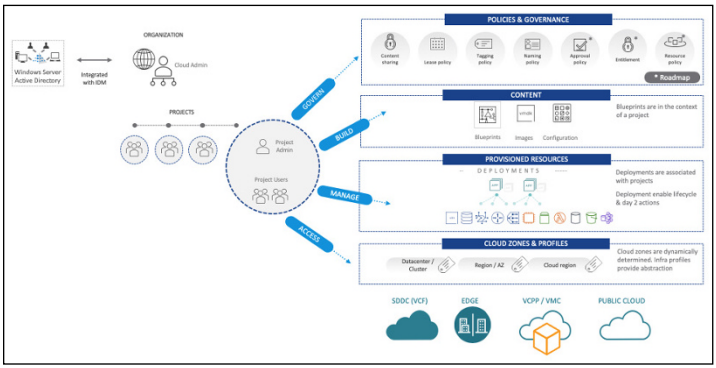


FIGURE 3-1: vRealize Automation organizations and projects provide governance over content, resources, and capacity.

Use of cloud zones

A Cloud Assembly cloud zone is a set of resources within a cloud account type such as vSphere or AWS. *Cloud zones* (refer to Figure 3-2) control where your blueprints deploy workloads. Each cloud zone is associated with projects and network profiles.

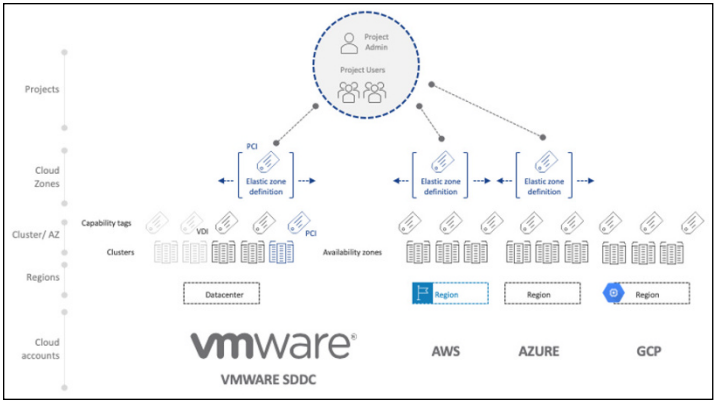


FIGURE 3-2: Cloud zones allow granular assignment of compute capacity for deployment needs.

Deployment of network services

Cloud Assembly integrates with NSX to deploy and use networks and load balancers. Provider infrastructure as code allows organizations to create blueprints that can be used to deploy on-demand resources. Alternatively, you can request networking services as part of an application deployment and the creation of these services rapidly without needing to understand NSX. Network admins can focus on other tasks without needing to manually configure network services as Figure 3-3 shows. You can also extend day-2 reconfiguration actions, reducing the administrative burden involved with supporting an environment.

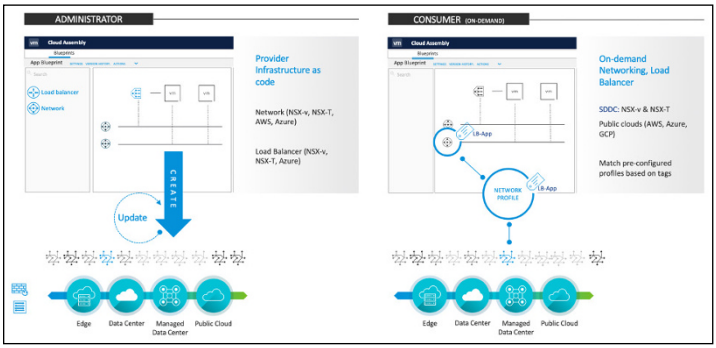


FIGURE 3-3: vRealize Automation supports the creation and management of networks and load balancers.

Security

Increasingly, applications must be deployed and secured as part of a single process. Cloud Assembly allows the creation of security groups and firewall rules as part of a deployment, as Figure 3-4 shows. Existing security groups may also be used. In both scenarios, virtual machines are assigned as security group members at deployment time.

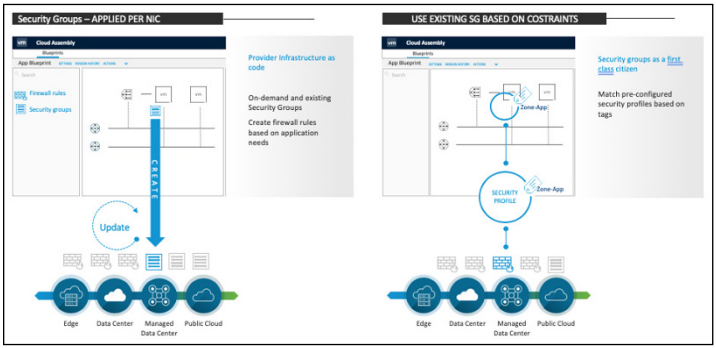


FIGURE 3-4: vRealize Automation supports the creation and management of security groups and firewall rules.

Tagging Objects

Cloud Assembly utilizes tags for placement determination and uses tags to constrain provisioning to specific compute, network, and security resources. Cloud Assembly calls these tags "capability or constraint" tags. You can add capability and constraint tags throughout Cloud Assembly to control placement. Other VMware products also use tags for organizational and informational purposes. Furthermore, Cloud Assembly can add tags to objects during a deployment; for example vSphere tags and NSX tags may be added.

REMEMBER

Adding tags is very straightforward in Cloud Assembly. In Figure 3-5, each time a segment or cloud machine is deployed from a configured blueprint, a tag is associated in NSX or vSphere. In the blueprint, `value:` is the tag in NSX, and `key:` defines the scope of the tag. After the tag is attached, both the scope and tag can be searched for in NSX or vSphere.

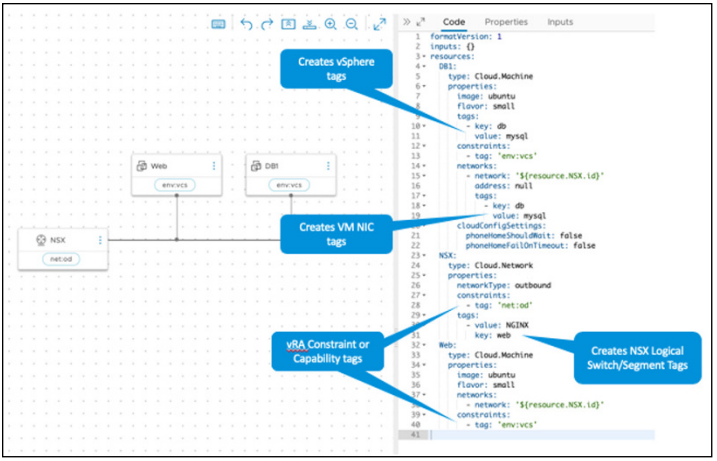


FIGURE 3-5: Tags are added to blueprints to control placement and for tag creation in NSX.

REMEMBER

To differentiate between tags in Cloud Assembly and NSX, keep the following in mind:

- Cloud Assembly uses constraint tags defined throughout the product to help the placement service make decisions about the best place to deploy requested resources.
- You add Cloud Assembly tags to a blueprint as well; however, Cloud Assembly tags appear as constraints, which Figure 3-5 shows. In this case, the tag constrains a deployment to a specific vSphere compute resource.
- NSX leverages object tags in a different way, mainly for third-party integrations, organizational, and management needs.

Working with Cloud Accounts and Compute Resources

To start, the first thing you'll do is configure Cloud Assembly to work with vSphere and NSX instances. These configurations are called *cloud accounts*, which Figure 3-6 shows. Upon completion, the vSphere and NSX instances will be linked in Cloud

Assembly. When you make deployment decisions involving networking, where the capabilities match the blueprint, you're taking into account the association between the vSphere and NSX cloud accounts.

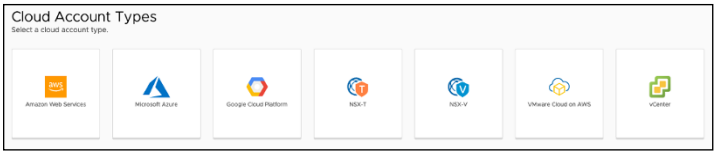


FIGURE 3-6: Supported cloud account types in Cloud Assembly.

After your vSphere and NSX accounts are configured, Cloud Assembly will display what's been discovered. Selecting Compute shows a list of discovered compute resources. Figure 3-7 displays how Cloud Assembly has discovered the cluster and resource pools from vSphere. Cloud Assembly uses these resources for Compute and Network placement. Before triggering a deployment in vRA, the ESXi Hosts must be configured to use NSX.

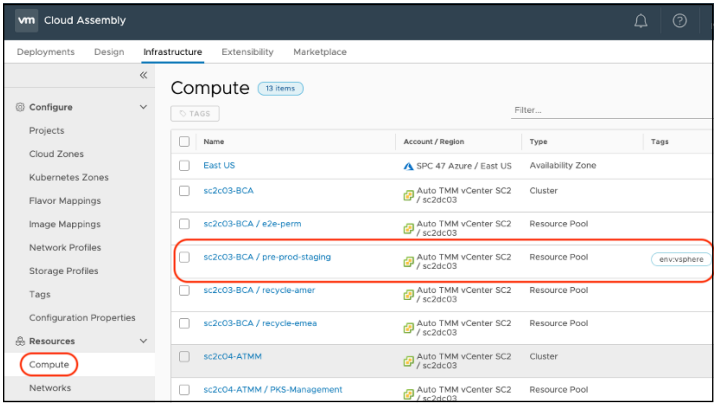


FIGURE 3-7: Discovered Compute resources after Cloud Accounts are configured.

Identifying the Network and Adding a Network CIDR

The Networks resource interface shows discovered logical switches from configured cloud accounts. Typically, you'll select the switch with an external route. Figure 3-8 shows the list of switches that have been discovered.

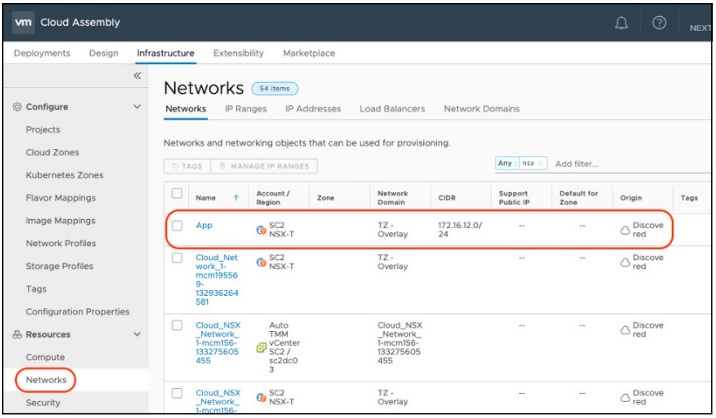


FIGURE 3-8: Existing switches configured in vSphere and NSX.

Clicking a switch name opens a new window where configurations are added and details shown for the switch. The first thing Figure 3-9 shows the network CIDR and default gateway have been added. Adding the CIDR and gateway allows the switch to be used later in the network profile for IP allocation and external access. Other details, such as domain, DNS servers, and search domains are also added through this interface. You can choose to make this the default switch for this zone. When a network isn't specified in the blueprint, the default switch is used when a blueprint calls for the switch's associated cloud zone compute resources.

In this case, add a tag key value pair, which allows you to use only this switch with VM deployments. After those areas have been configured, you can focus on setting up network profiles.

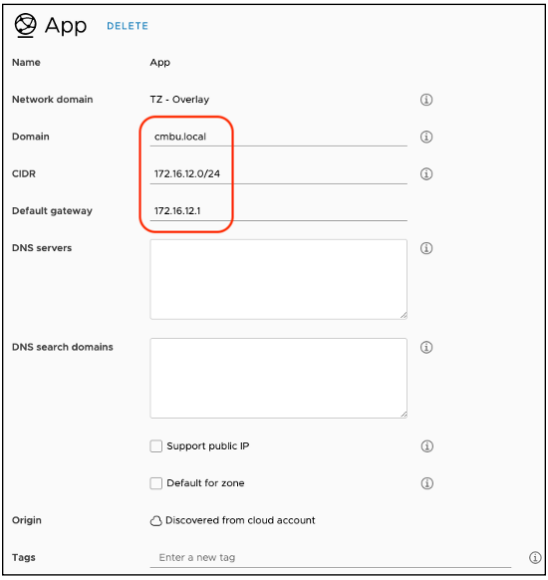


FIGURE 3-9: A discovered switch configured for later use in a network profile.

Network profiles control the network constructs and configurations that are used for placement decisions during a deployment. They also control the level of isolation a workload will have when deployed. The first network types we cover are existing networks and outbound, also known as *on-demand networks.* Before diving into network profiles, you need to understand infrastructure-as-code blueprints and their importance in the network automation realm.

Working with infrastructure-as-code blueprints

The best place to begin discovering how Cloud Assembly interacts with networking is by examining a blueprint. Figure 3-10 shows a Single-VM-Nat. As the blueprint name suggests, a single machine object and network with a NAT rule is created, among other configurations, during the deployment.

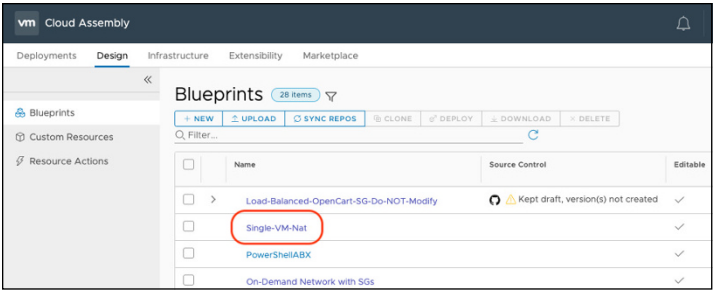


FIGURE 3-10: A Single-VM-Nat blueprint.

Figure 3-11 selects an NSX network entity and connects it to a cloud machine. To connect a network and VM, hover over the cloud machine on the blueprint canvas, click the cloud machine, and then drag the cursor to the Network object. After doing this, you'll see a connection line between the two objects. Each network object you connect to a cloud machine will create a separate NIC at deployment time. The YAML code will update to include the connection between these objects.

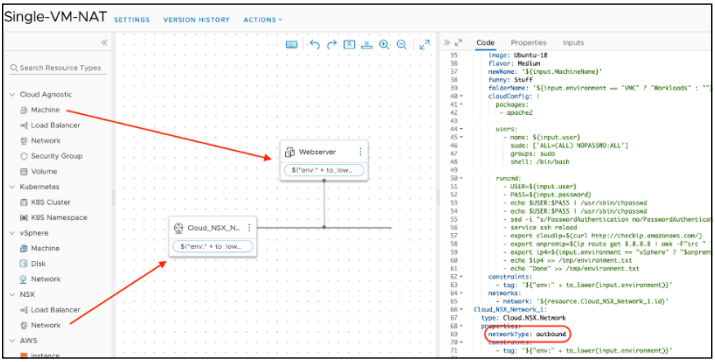


FIGURE 3-11: Simply drag and drop desired network and machine objects to the blueprint canvas.

On the YAML portion of the blueprint, the *networkType: outbound* property creates a new network with outbound access and NAT configured. The YAML code includes a networkType setting, which instructs the placement engine to look for a network profile that matches outbound.

Removing outbound from the YAML panel displays available networkType options as in Figure 3-12. In the next sections, we show you how to configure Cloud Assembly network profiles to accommodate common blueprint settings.

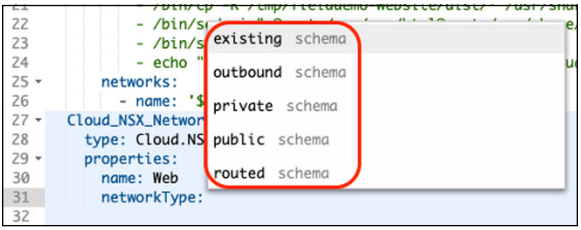


FIGURE 3-12: The list of available network types for an NSX network object.

Network origin and networkType: existing

networkType: existing uses one of the following:

- **Discovered origin:** Manually created network objects found through the Cloud Assembly discovery service
- **Deployed origin:** Network objects provisioned by Cloud Assembly

For example, a provider network and previously provisioned on-demand network would appear as deployed. A *provider network* is created from a blueprint where the only object on the canvas is a network; no virtual machines are associated. An organization may want to create a static provider network for developers to use without creating an on-demand network each time a deployment occurs in certain scenarios. The key thing to remember is both Origin types can be used for existing network types.

REMEMBER

When NSX is involved, deployed, and discovered, networks can be a switch configured for VXLAN or Geneve overlay networking. The switch could be VLAN backed too. Workloads deployed, using *networkType: existing*, with either origin type, use static or dynamically assigned IP addresses depending on your NSX, VM, and configuration management options.

Switching from this blueprint to a network profile, you can add existing networks that will be used for placement decisions. Choose

App and DB with constraint tags for each switch. Constraint tags allow you to assign specific networks to each tier of the application in the blueprint. The switches have their own DHCP server and assigned IP range. Remember, for existing networks, DHCP is configured in NSX (or third-party DHCP services) independently from Cloud Assembly. Use the configuration as shown in Figure 3-13. App and DB will have their own distinct IP ranges.

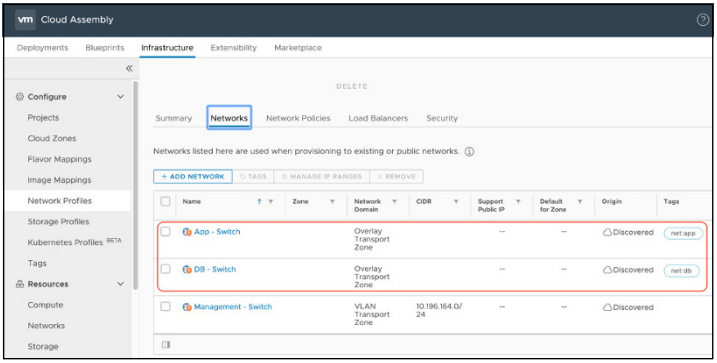


FIGURE 3-13: For existing networks, compute workloads are assigned to switches defined in the network profile.

Next, navigate to the network policies option and confirm *Do not create on-demand network or on-demand security group*, as shown in Figure 3-14.

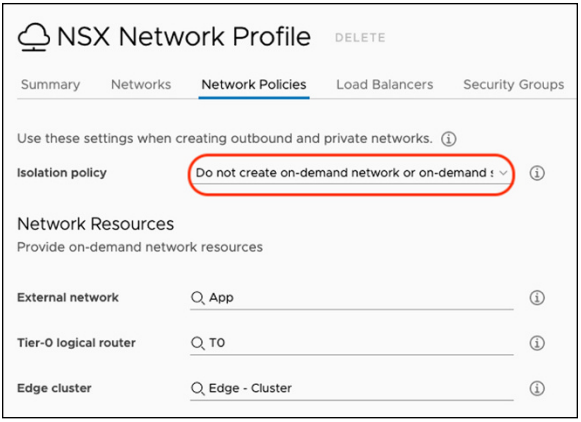


FIGURE 3-14: The network policies tab controls the type of network a network profile will configure.

networkType: outbound

networkType: outbound creates on-demand networks. By default, on-demand networks use a mixed DHCP and static IP range assignment. As shown in Figure 3-15, when creating an on-demand network with NSX, each blueprint deployment creates a new T-1 gateway, L2 switch, one-to-many SNAT rule, DHCP server with IP pool, NAT route advertisement, and the proper uplinks/downlinks. Allocated static IPs and DHCP IP pools are based upon the CIDR and subnetting configuration specified in the network policies and network options.

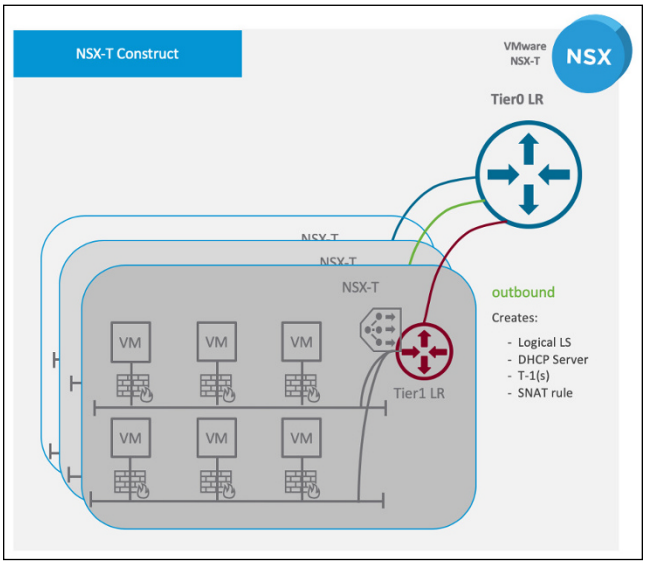


FIGURE 3-15: Network configuration diagram for outbound networks.

Manage IP Ranges (see Figure 3-16) creates a pool of IPs that Cloud Assembly will allocate for use as translated external IPs. The IPs are assigned to each SNAT rule that is created in NSX. The addresses are also used for Load Balancer VIPs and static IP assignment, if you choose to deploy load balancers or assign static IPs.

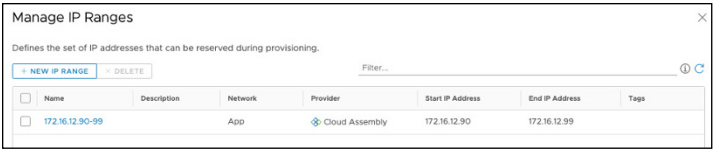


FIGURE 3-16: Create and manage IP ranges for each switch.

Cloud Assembly tracks the allocation of IP addresses using a built-in IPAM capability (refer to Figure 3-17). Click the IP range name to view allocation details. If you're running out of IP addresses, simply modify the start and end IP addresses to increase the pool size. Alternatively, you can add a new IP range. You can use Infoblox IPAM to manage and track IP allocation as well.

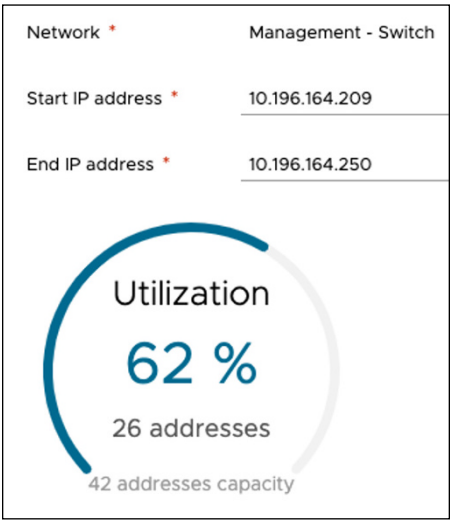


FIGURE 3-17: Track IP allocation for the configured IP range.

In a network profile, switching to the network policies option and selecting *Create an on-demand network* sets the profile for outbound network types. You'll need to add a desired network CIDR, subnet size, and IP range assignment. Figure 3-18, under IP Address Management, chooses /28, which means a separate /28 subnet, based on the defined CIDR, will be assigned to each on-demand network that is created.

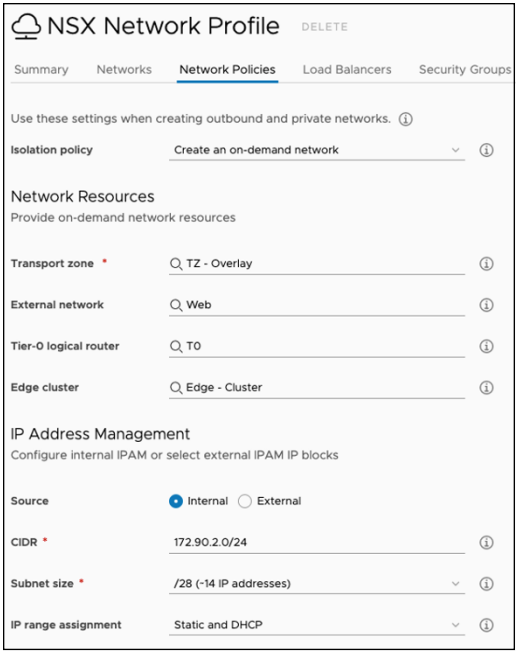


FIGURE 3-18: Use the network policies tab to define subnets.

Clicking the subnet size dropdown presents the available subnet options as in Figure 3-19. Subnet size instructs Cloud Assembly to create a new network based on a portion of the defined CIDR — in this case part of 172.90.2.0/24. Each /28 network Cloud Assembly adds to NSX, creates a T-1 gateway, logical switch, DHCP server and pool, route advertisement, and SNAT rule. VMs are assigned to the switch as part of the deployment process.

IP range assigment controls how Cloud Assembly assigns IP addresses to VMs. The default is Static and DHCP (or mixed) when a new profile is created. The Static and DHCP setting instructs Cloud Assembly to create two IP ranges. In our example, we use a /28, so two /29 s will be created for IP assignment. Where applicable, one /29 will be used to assign static IP addresses to VMs and the other for dynamic assignment to VMs. All this happens behind the scenes; you don't need to worry about further configuration in

Cloud Assembly. You can also choose either Static or DHCP from the dropdown menu as in Figure 3-20. Doing so creates only one range, and IPs are assigned based on either setting.

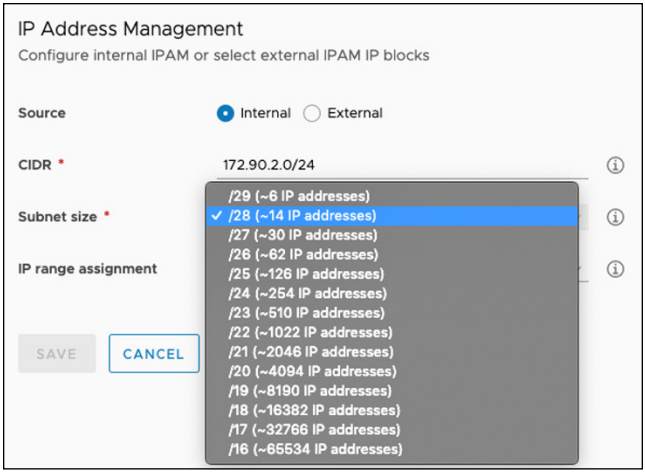


FIGURE 3-19: Choose the subnet size for each deployment.

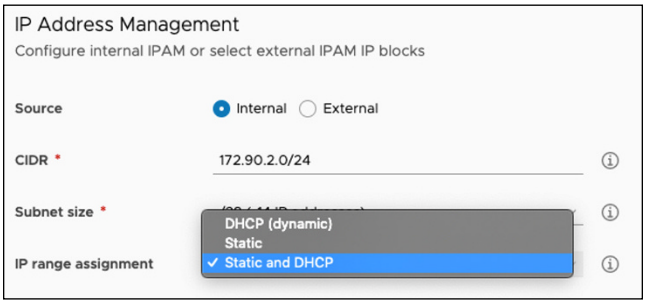


FIGURE 3-20: Choose an IP range assignment.

Choose an IP assignment type

The blueprint controls whether a VM receives a static or dynamic IP. Add assignment: static to the YAML network properties of a VM in the blueprint to assign a static address as in Figure 3-21. If assignment: static isn't present, a dynamic IP address will be used.

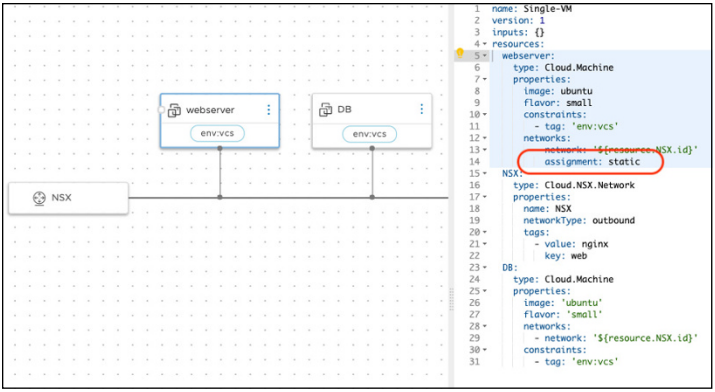


FIGURE 3-21: Adding assignment: static to the blueprint.

Examining NSX Security Groups

This section looks at how vRealize Automation works with NSX security groups. Security groups are one method used to isolate VMs constrained to private network type deployments. Security groups and associated firewall rules are a core way you can implement and manage security with NSX. Cloud Assembly can assign VMs to existing NSX security groups through network profiles. Figure 3-22 shows how an existing security group is added to the profile. You can also assign membership to multiple security groups using this network profile.

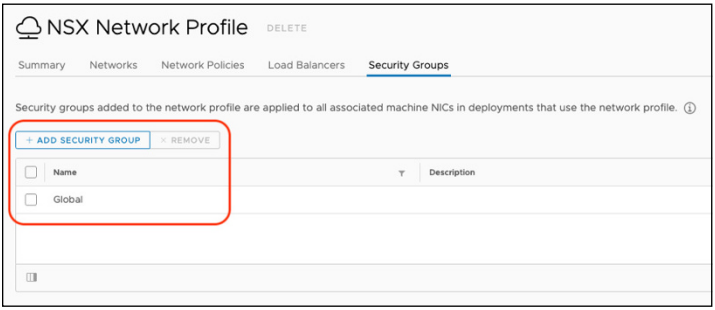


FIGURE 3-22: Add security groups to a deployment.

Security groups are visible within Infrastructure ➪ Resources ➪ Security. The view in Figure 3-23 allows you to see all the discovered security groups. You can also choose a security group, click TAGS, and add a constraint tag for later use with a blueprint. Constraint tags must be added to an existing Security Group to assign VM membership during deployment.

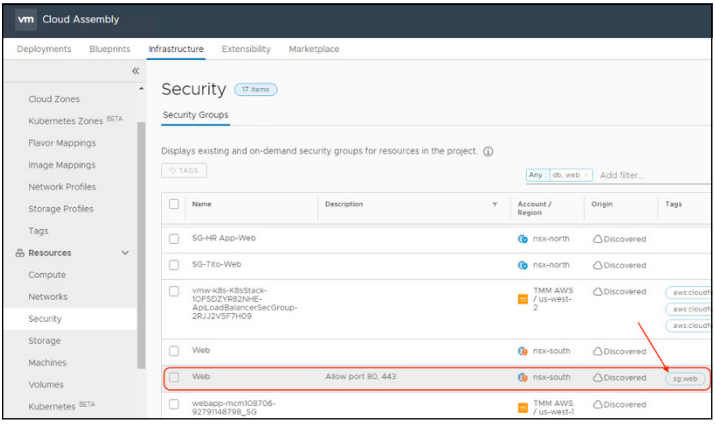


FIGURE 3-23: Assign tags to security groups through Resources – Security.

Security groups can be added to the canvas from the component list. Two types of security group objects can be specified in a blueprint: existing and new. We cover how to define both types in the blueprint in the following text.

Adding Existing Security Groups via Blueprint

To add an existing security group via the blueprint, click-drag the object to the canvas. You'll notice the YAML code updates with information on each security group that's added. Figure 3-24 shows an example of two existing security groups attached to VMs.

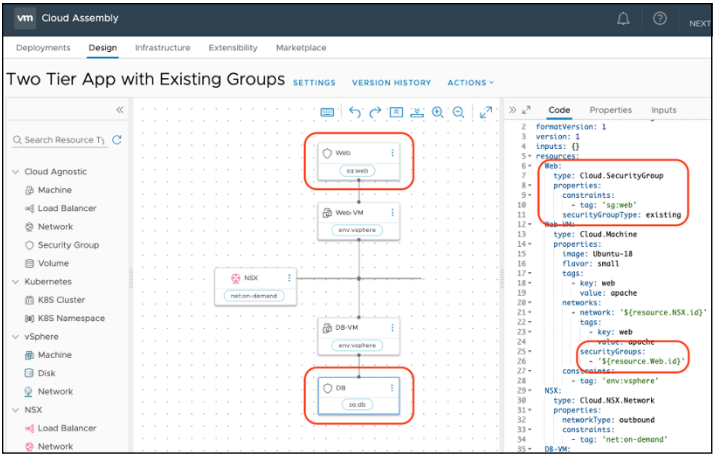


FIGURE 3-24: Existing security groups can be added directly from the blueprint canvas.

To associate a security group with a cloud machine, click the security group object on the canvas and drag the cursor to the cloud machine (similar to the process used to connect machine and network objects). A dialog will appear asking which NIC to assign the security group to. Don't forget to add the security group constraint tag to the YAML portion of the blueprint as well. As soon as the machines deploy, they'll become members of the specified existing security groups.

When a security group, whether existing or new, is connected to a cloud machine, a resource binding appears in the YAML code for the connected cloud machine. You can assign independent security groups to each NIC, if so desired.

REMEMBER

Assigning security groups from the blueprint offers more flexibility than assigning via the network profile. However, both methods can still be used for deployments. You may want to assign a common security group, which sets a minimum security baseline, at the network profile and more secure or specific security groups using the blueprint.

Adding On-Demand Security Groups to a Blueprint

Cloud Assembly provides the option to choose "new" for securityGroupType in the YAML code. You also have the option to create firewall rules and take advantage of existing services in NSX. Figure 3-25 shows how to define source; destination; direction, inbound or outbound; ports; protocol; existing service; and source.

A security group with associated firewall rules is created during the deployment assigned to the NICs of VMs created during the deployment. The security group and rules are removed when the deployment is deleted. The process for connecting the security group and cloud machine are the same as existing groups; here though you won't use a constraint tag in the YAML code because in this example we're creating new security groups.

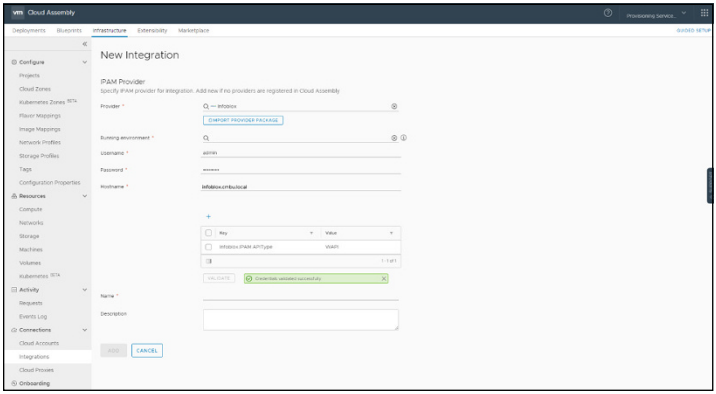


FIGURE 3-25: Firewall options and new designation in the YAML code.

Figure 3-26 shows new security groups being created in NSX. The VMs, Web and DB, will be members of the newly created groups with the distributed firewall rules set in the YAML code. You can also set the source as other VMs in the deployment or a range of IP addresses.

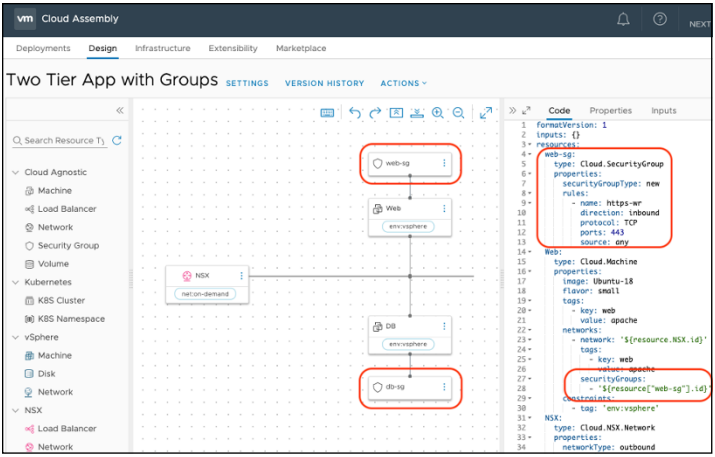


FIGURE 3-26: YAML options for controlling firewall rules and resource bindings.

Private network types

The private network type (*networkType: private*) isolates provisioned VMs from external access via firewall rules or network configuration. Private allows you to use existing networks, as shown in Figure 3-27, or on-demand networks for deployments. As with other network types, the network profile primarily controls the deployment configuration. If you choose private on the blueprint and use a constraint tag for a network profile/policy where *Create an on-demand security group* is matched, a security group is created for each network on the blueprint canvas. All on-demand security groups are created with three associated firewall rules, which are as follows:

- **Rules 1 and 2:** Inbound and outbound reject rules
- **Rule 3:** Intra-VM communication for members of that security group

REMEMBER

If you choose to create an on-demand private network, you'd still use networkType: private on the blueprint, but you'd constrain the network choice to a network profile with on-demand networking configured. In that event, a DHCP server and pool are created; however a T-1 isn't configured, and no security groups are created.

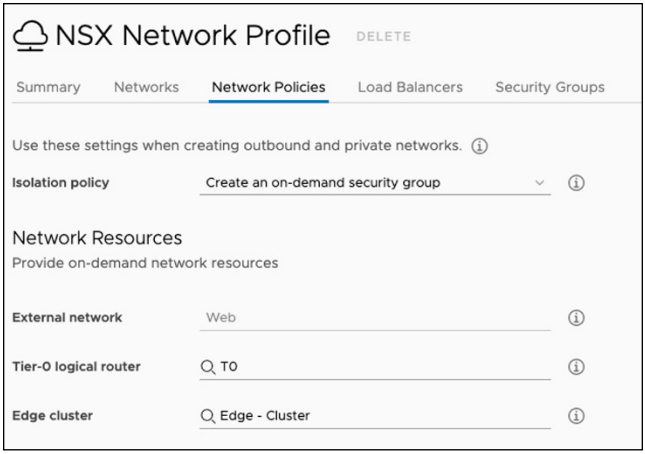


FIGURE 3-27: Configuration of private networks with security groups.

Private networks may use one of two configuration options in the network profile and network policy tab:

- For a private **on-demand security group** deployment, add an existing deployed or discovered switch in Networks, choose *Create an on-demand security group* in network policies, and specify *networkType:private* on the blueprint.
- For a private **on-demand network deployment**, the blueprint still uses private; however, from there the configuration process uses the on-demand (outbound) network profile config. The main difference with this configuration is you don't need to add an existing switch in Networks and an external network isn't required in network policies.

Routed network types

networkType: routed is only available for NSX and requires an NSX network object on the blueprint canvas. You can't use the routed network type with a cloud agnostic network object because the option won't appear in the YAML properties. Routed networks are similar to outbound (on-demand networks), but they configure the route advertisement to advertise all connected routes for the created Logical Router (Outbound uses Advertise on all NAT routes), and they don't create a NAT rule.

Simply point a blueprint with networkType: routed to an on-demand network profile/policy (just like the outbound type) and Cloud Assembly will handle the configuration at deployment time. Refer to Figure 3-28 for an example.

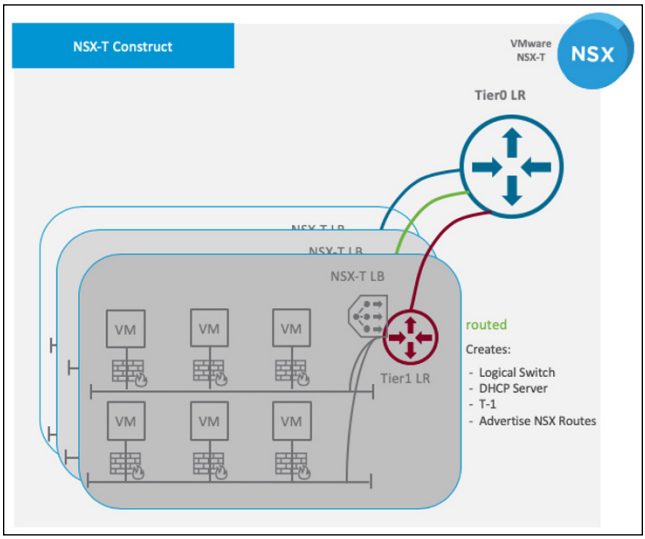


FIGURE 3-28: Network configuration diagram for routed networks.

On-demand load balancers

You can create on-demand load balancers with or without attached VMs. *Load balancers* are defined within a blueprint and can be used for other deployments similarly to how on-demand networks can be used. At deploy time, a load balancer, virtual server, server pool, and monitor are added to NSX except with provider load balancers. A load balancer created without an attached VM in the blueprint is called a *provider load balancer.* When a provider load balancer is created in NSX, the virtual server, server pool, and monitor aren't created until VMs are associated.

Figure 3-29 shows a blueprint configured to create a provider load balancer. You can add either a cloud agnostic load balancer or NSX load balancer to the canvas. The main difference is an NSX load balancer supports UDP in addition to TCP. The load balancer configuration is highlighted on the YAML portion of the blueprint in this figure.

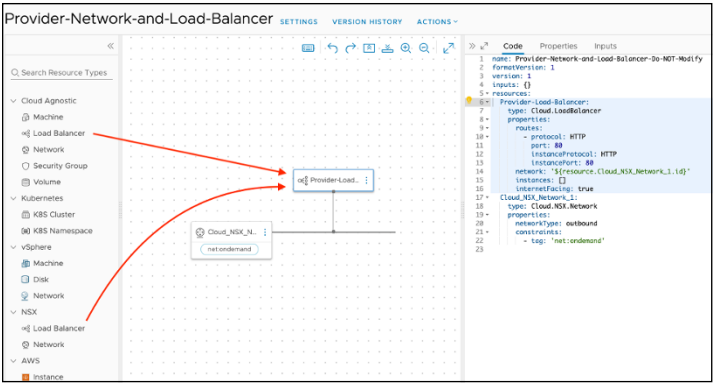


FIGURE 3-29: Provider network and load balancer deployment.

Existing load balancers

Existing load balancers are added from the network profile that is used for the deployment. Within the network profile choose Load Balancers ➪ Add Load Balancer and select the discovered or deployed load balancers in your environment. You can also use constraint tags to determine the load balancer. If a load balancer is selected in the network profile (see Figure 3-30) and defined in the blueprint (shown in the previous section), a new virtual server is created for the existing load balancer using the specified YAML configuration values.

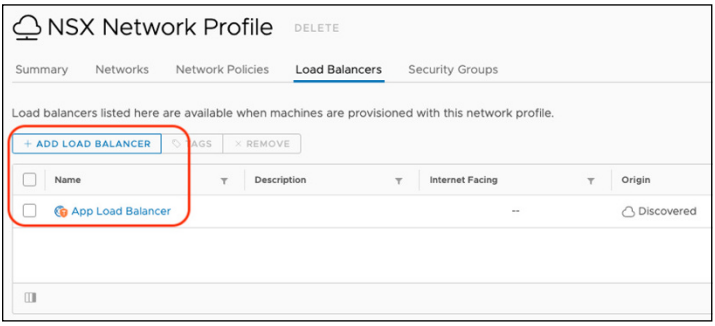


FIGURE 3-30: Adding an existing provider load balancer in the network profile.

TIP

You can view all load balancers that are discovered or deployed within the Infrastructure ➪ Resources ➪ Networks ➪ Load Balancers tab (refer to Figure 3-31). Using this interface, you can find out more about the load balancers that are under Cloud Assembly management, including Cloud Account/Region, whether the load balancer is Internet facing, the Origin, and associated constraint tags.

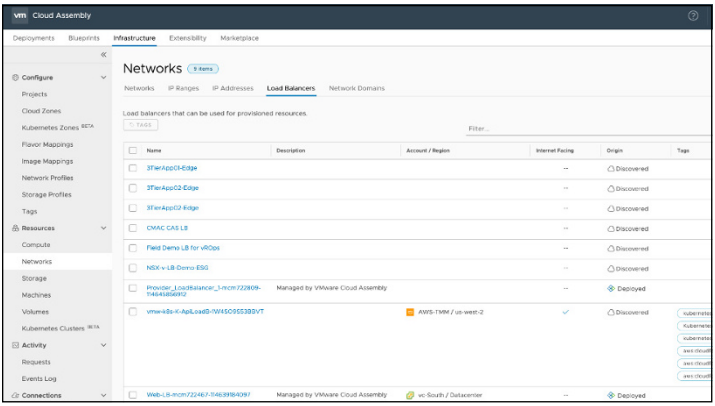


FIGURE 3-31: Discovered load balancers in the Resources view.

REMEMBER

Selecting a load balancer name opens a details page (see Figure 3-32). The details page provides further information on IP address, usage, routes, custom properties, health check configuration, and the ability to add a constraint tag.

Day-2 reconfiguration of load balancers

Cloud Assembly also supports reconfiguring load balancers as a day-2 action. To reconfigure a load balancer, navigate to the Deployments screen, select a deployment that includes a load balancer, click the load balancer in the Topology view, and then click the Actions dropdown and select Reconfigure as Figure 3-33 shows. You can add or modify protocol, port, and health check configuration.

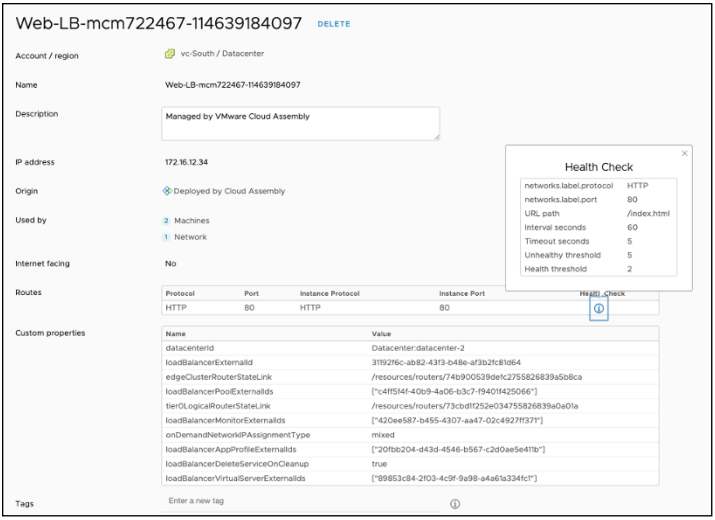


FIGURE 3-32: Load balancer configuration details.

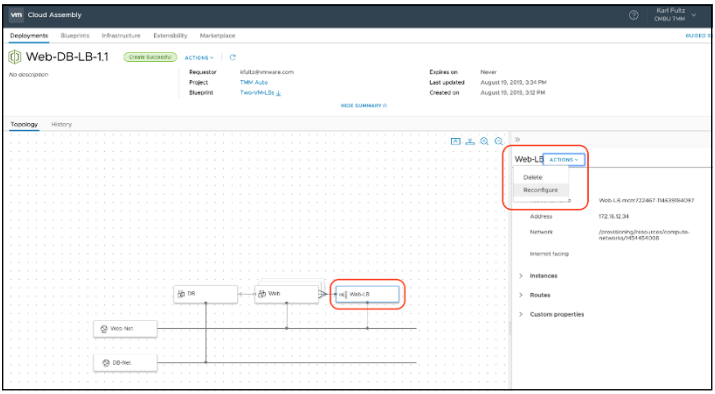


FIGURE 3-33: Day-2 reconfiguration of the load balancer.

IPAM – Infoblox support

Cloud Assembly includes integration with Infoblox IPAM solutions within your environment, which is shown in Figure 3-34. After the integration configuration is complete, you're able to provision IP addresses using defined networks in Infoblox. Assigned IP ranges from Infoblox are added and made available for deployments

through the Cloud Assembly network profiles – network policies under IP Address Management – External (as shown in Figure 3-18). Infoblox IP assignment configuration in the Cloud Assembly network profile is similar to the built-in IPAM configuration. After you're familiar with configuring IP Address Management in network profiles – network policies, the external IPAM process will be an easy transition.

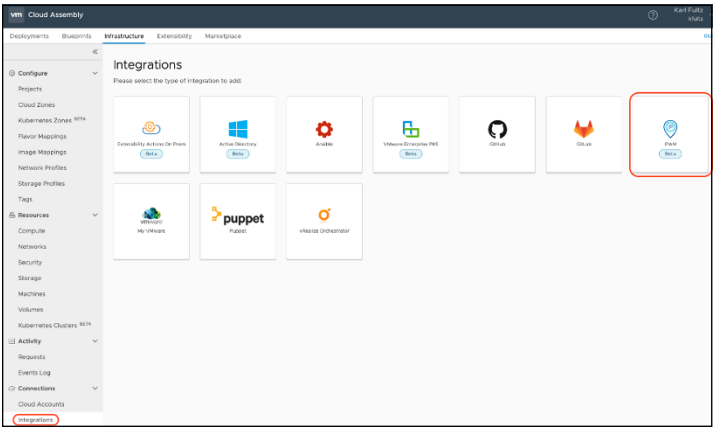


FIGURE 3-34: Add an IPAM provider to Cloud Assembly.

IP assignment type and specific IP addresses are defined within the Cloud Assembly blueprint. Machines deployed from Cloud Assembly have a MAC address and IP statically or dynamically assigned. All of these configurations are tracked within Infoblox. IP range details are also tracked within Cloud Assembly through Resources ➪ Networks following deployment. Additionally, Cloud Assembly will update DNS records in Infoblox with the machine name created during the deployment. After a deployment is deleted from Cloud Assembly, the IP allocation and DNS record is removed from Infoblox. Figure 3-35 shows how to configure the Infoblox integration within Cloud Assembly.

With external IPAM support and an available IPAM SDK, Cloud Assembly offers powerful IP address management for your compute and networking deployments. For a deeper dive into the IPAM integrations with Infoblox, take a look at this whitepaper from VMware's Engineering team.

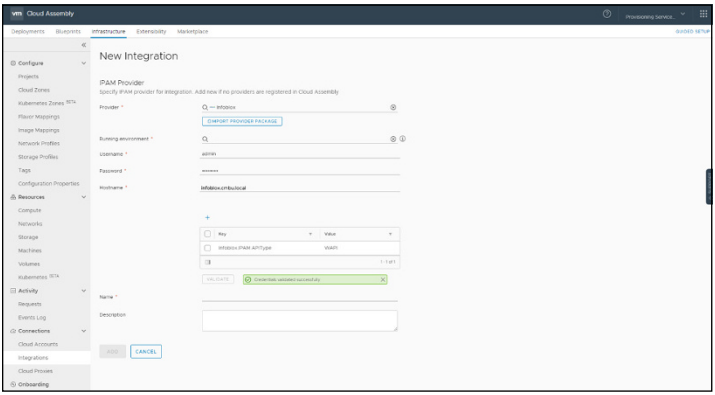


FIGURE 3-35: Configuring the IPAM integration.

Releasing Blueprints to the Service Broker Catalog

After Cloud Assembly is configured and your blueprints are complete, the next step is to release the blueprints to the service broker catalog (refer to Figure 3-36). Whereas Cloud Assembly is primarily meant for cloud administrators, vRealize Automation – Service Broker is the service most users will interact with to request deployments. Think of this catalog as a storefront for users to request and consume resources. Service Broker also allows cloud administrators to apply a deployment lease, allowed day-2 actions, and approval requirements for each request.

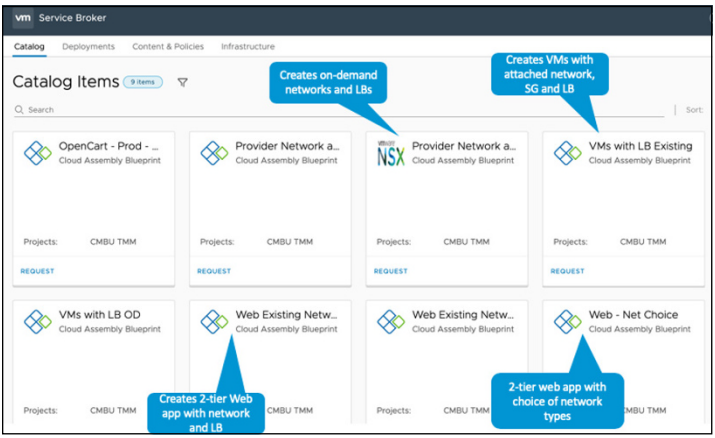


FIGURE 3-36: Request deployments from the Service Broker catalog.

IN THIS CHAPTER

- Comprehending available SDKs
- Generating your own SDK
- Looking at opensource tools like Ansible and Terraform
- Examining PowerCLI

Chapter 4
Utilizing SDKs and Tools

Sometimes you need a more traditional approach to automation, perhaps because of an already existing domain expertise on a certain programming language or because it's easier to leverage existing resources. Whatever the reason, NSX does provide a way to write automation tools using standard programming languages through SDKs. This chapter explains various tools like SDKs available for NSX and gives an overview on using tools like PowerCLI, Ansible, and Terraform with NSX.

Understanding Available SDKs

You can access generated SDKs, which are available for direct download at `downloads.vmware.com`. The API spec generates these SDKs so each time a new API is introduced or an API changes, the SDKs automatically reflect them. Here are some standard SDKs and a quick peek into using them.

Using Python SDKs

Using the Python SDK is as simple as using any other Python library. You first make a connection to the NSX manager and use

that connection to create other objects. The following example shows using Python SDK to connect to NSX and performing a task:

```
# Connect to NSX-T Mgr
session = requests.session()
session.verify = False
nsx_url = 'https://%s:%s' % ("192.168.22.32", 443)
connector = connect.get_requests_
  connector(session=session, msg_protocol='rest',
  url=nsx_url)
stub_config = StubConfigurationFactory.
  new_std_configuration(connector)
security_context = create_user_password_security_
  context("admin", "VMware1!")
connector.set_security_context(security_context)
transportzones_svc = Transportzones(stub_config)
```

Using Java SDKs

Utilizing Java SDK is similar too. You create an API client and then invoke the required method. Here is an example of creating a transport zone:

```
TransportZone transportZone = new TransportZone.
  Builder(
    TransportZone.TRANSPORTTYPE_OVERLAY)
    .setDisplayName("My Transport Zone")
    .setDescription("Transport zone for demo")
    .setHostSwitchName("hostswitch1").build();
TransportZone resultTZ = zoneService.
  create(transportZone);
```

You can find the full API documentation at `https://downloads.vmware.com`.

Focusing on Opensource Tools

NSX also supports opensource tools like Ansible modules and Terraform provider, which we discuss in greater detail here.

NSX Ansible modules

NSX Ansible modules are available for you to use from any of these places:

- https://github.com/vmware/ansible-for-nsxt
- https://my.vmware.com/en/web/vmware/info/slug/networking_security/vmware_nsx_t_data_center/2_x#drivers_tools

The NSX Ansible module is fully open sourced, and you can download it from either GitHub or https://downloads.vmware.com/. The Ansible modules provide a way to fully install, configure, and upgrade NSX systems. You can also use these modules to create a full end-to-end automation along with other popular modules like *vmware_guest*.

The online modules come with their own set of documentation and working examples. Entire workflows are available that create the NSX manager cluster, deploy transport nodes, and configure the logical objects.

Ansible itself is available as a Python package and can be installed using *pip*. After they're installed, you can use the modules to deploy a fully functional production–ready NSX environment. Ansible modules accept parameters that let users customize the environment. You can just edit these parameters and start using existing modules.

Here is a simple Ansible playbook that can create segments on NSX:

```
---
#
# Playbook to create Segment
#
- hosts: localhost
  become: yes
  tasks:
    - name: Create Segment
      nsxt_segment:
        hostname: "192.168.22.32"
        username: "admin"
```

```
        password: "myPassword1!"
        validate_certs: False
        display_name: west-Segment-4
        state: present

        transport_zone_display_name:
"transportzone-730"
        subnets:
          - gateway_address: "40.1.1.1/16"
        segment_ports:
          - display_name: seg-port-4-11
            state: present
```

Using the exact same playbook, you can create and delete objects. In the case of NSX Ansible modules, each module supports a *state* property. If the state is *present*, then the object is created. Changing the state to *absent* and running the exact same playbook deletes the object. It's that simple.

Check out the modules and examples available on GitHub and contribute. You can also read periodic blogs published on `http://blogs.vmware.com` that address the specific topic of automating NSX with Ansible.

Realize Automation also integrates with Ansible Open Source to trigger configuration tasks as part of a deployment workflow. Ansible Open Source playbooks are a 1stclass citizen on the blueprint canvas in vRealize Automation – Cloud Assembly. Playbooks are triggered through direct calls to the Ansible Control Host at deployment time. The Ansible integration with vRealize Automation adds critical elements of governance, visibility, and simplicity to the equation, which Ansible alone doesn't provide to administrators and end users.

NSX Terraform provider

The NSX Terraform provider gives the administrator another way to manage the NSX infrastructure. You can download the provider from any of the following places:

- `www.terraform.io/docs/providers/nsxt/`
- `https://github.com/terraform-providers/terraform-provider-nsxt`

» https://my.vmware.com/en/web/vmware/info/slug/networking_security/vmware_nsx_t_data_center/2_x#drivers_tools

Installing the NSX Terraform provider depends on where you get the sources. The easiest would be to do a *terraform init,* which would install the provider and dependencies from the Terraform.io site. In case the latest developer code is required, then you can download the sources from GitHub or the VMware site and compile them.

NSX Terraform provider has support to create complex configurations that include setting up routing, switching, and setting up distributed firewall rules.

After you install it, Terraform operates through a manifest file that is quite declarative. Here's an example of creating a new NSX segment:

```
provider "nsxt" {
   host                        = "192.168.22.32"
   username                    = "admin"
   password                    = "myPassword1!"
   allow_unverified_ssl        = true
}

resource "nsxt_policy_segment" "test" {
  count = 2
  display_name = "demo-${count.index}"
  connectivity_path = nsxt_policy_tier1_gateway.
   demo.path

  subnet {
     cidr = "2.2.${count.index}.1/24"
  }

  tag {
    scope = "color"
    tag   = "orange"
  }
```

```
    advanced_config {
      connectivity = "ON"
      local_egress = false
    }
  }
```

REMEMBER

Working with Terraform typically involves three clear steps:

1. **Create the configuration file, which defines the desired state.**
2. **Run *Terraform Plan* to determine what actions are necessary to achieve the desired state specified in the configuration files.**
3. **Run *Terraform Apply* to actually apply the changes.**

You can delete created items using *Terraform Delete* command as with any inventory management system.

Although there are ways to import existing objects into the Terraform inventory, Terraform works best if it's the only source of truth. Working with Terraform becomes a breeze if you create, edit, and delete all objects through Terraform.

Support structure

Both NSX Ansible modules and NSX Terraform providers are opensource tools. Unlike traditional opensource tools that are completely community driven, NSX Ansible modules and NSX Terraform provider have a mixed mode of operation:

- Sources are available on GitHub for anyone to use and contribute.
- Both the NSX Ansible module and Terraform provider are fully supported. Yes, you can call VMware support and get help. Just make sure you use the right branch or version.
- Because the repositories on GitHub are public, you can use, contribute, or raise feature requests or issues.
- The latest code development happens on GitHub, and yes, you can use it, and with the best-effort community support.
- Best of all, it's completely free.

Whether you're a new NSX user looking at automation options or a large enterprise team with dedicated automation resources, NSX Ansible modules and Terraform provider give flexibility and provide easy adoption of NSX.

Inspecting NSX PowerCLI

NSX Automation landscape isn't complete without addressing PowerCLI for Windows PowerShell users. VMware PowerCLI provides a command-line scripting tool for interacting with NSX-T. Check out `https://code.vmware.com/web/tool/11.5.0/vmware-powercli` to understand more about PowerCLI.

NSX-T PowerCLI works similar to Java or Python SDK. You first make a connection and then interact with the NSX objects.

Here's an example of making the initial NSX connection request and creating a Tier1 Gateway:

```
#Connect to NSX Manager
Connect-NsxtServer -Server 192.168.22.32 -User
  admin -Password myPassword1!
#Retrieve Router Information
$t1routerdata = Get-NsxtPolicyService -Name com.
  vmware.nsx_policy.infra.tier1s
#Set Variables
$t1routerspecification = $t1routerdata.Help.patch.
  tier1.Create()
$t1routerspecification.description = "Created with
  PowerCLI"
$t1routerspecification.id = "MyNewTier1"
$t1routerspecification.display_name = "My New
  Tier1"
$t1routerspecification.tier0_path = "/infra/
  tier-0s/myTier0"
$t1routerspecification.route_advertisement_types =
  @("TIER1_IPSEC_LOCAL_ENDPOINT,
  "TIER1_CONNECTED")
#Add Tag to the Router
$t1routertag = $t1routerdata.Help.patch.tier1.
  tags.Element.Create()
```

```
$t1routertag.tag = "powercli"
$t1routerspecification.tags.Add($t1routertag) |
  Out-Null
#Create T1 Router
$t1routerdata.patch($t1routerspecification.id,
  $t1routerspecification)
```

You can download the NSX PowerCLI SDK from the main VMware PowerCLI page at `https://code.vmware.com/web/tool/11.5.0/vmware-powercli`.

IN THIS CHAPTER

- Checking online for a wide array of options
- Watching videos and webinars

Chapter 5
Ten Resources to Help You Get Started with Network Automation

This book presents an introduction to network automation and why it's an important topic for your organization. If you want to take a deep dive into it, we're here to help. This chapter presents a list of ten resources and tutorials to enhance your understanding of network automation and help you get started.

Websites

You can find a plethora of websites with helpful information. We suggest you begin with the following to complement this handy guide:

- **NSX REST APIs** (`https://code.vmware.com/apis/892/nsx-t`)
- **NSX Terraform Provider** (`www.terraform.io/docs/providers/nsxt/index.html`)

- **VMware** (www.vmware.com/solutions/network-automation.html)
- **vRealize Automation APIs** (https://code.vmware.com/apis/vrealize-automation)
- **vRealize Automation IPAM SDK** (https://code.vmware.com/sdks)
- **Ansible for Network Automation** (https://docs.ansible.com/ansible/latest/network/getting_started/index.html)
- **Ansible Modules for NSX-T** (https://github.com/vmware/ansible-for-nsxt)
- **Batfish** (www.batfish.org)
- **Jinja2** (https://jinja.palletsprojects.com/en/master/templates/)
- **Napalm** (https://napalm-automation.net/)
- **Nornir** (https://nornir.readthedocs.io/en/latest/#)
- **SaltStack for Network Automation** (www.saltstack.com/solutions/netops)

Discussion Groups

You can join a discussion group to post questions and connect with your fellow network automation engineers in the industry. Here are some discussion groups we recommend:

- **Ansible Community** (www.ansible.com/community)
- **Network to Code Slack** (http://slack.networktocode.com/)
- **Terraform Discussion** (https://discuss.hashicorp.com/c/terraform-providers/vmware/39)
- **VMware{code}** (https://code.vmware.com/)

Analyst Research

Experts in the field of network automation are a great resource. Get an independent analyst's view on the state of network automation via these resources:

- **Three Ways to Improve Network Automation, Gartner** (subscription, ID G00382803)
- **How to Automate Your Network Using DevOps Practices and Infrastructure as Code, Gartner** (subscription, ID G00407239)
- **Jump-Start Your Network Automation, Forrester** (subscription)
- **Jump-Start Network Automation to Scale Digital Initiatives, Gartner** (subscription, ID G00322992)
- **NetOps 2.0: Embrace Network Automation and Analytics to Win in the Era of ContinuousNext, Gartner** (subscription ID G00390284)
- **Networking and DevOps, Gartner**

Books

When you're ready to take a deeper dive into network automation, why not get the blueprint from the technical experts to help you understand what's going on "under the hood"? Here are some of our book recommendations:

- *Automate Your Network: Introducing the Modern Approach to Enterprise* Network Management by John W. Capobianco (self-published)
- *Go Programming for Network Operations* by Tom McAllen (self-published)
- *Learning Ansible: Automate Cloud, Security and Network Infrastructure Using Ansible 2.x* by Russ McKendrick (Packt Publishing)
- *Mastering Python Networking* by Eric Chou (Packt Publishing)
- *Network Automation Using Python 3* by Jithin Alex (self-published)

- *Network Programmability and Automation* by Jason Edelman, Matt Oswalt, and Scott A. Lowe (O'Reilly Media)
- *Network Programmability with Yang* by Jan Lindblad, Benoit Claise, and Joe Clark (Addison-Wesley Professional)
- *Practical Network Automation* by Abhishek Ratan (Packt Publishing)

Blogs/Publications

Many network automation experts are blogging about the lessons learned and sharing example workflows. Follow their conversations on blogs and Twitter:

- **VMware Cloud Management blogs** (`https://blogs.vmware.com/management`)
- **VMware Network Virtualization blog** (`https://blogs.vmware.com/networkvirtualization/`)
- **Getting started with the vRealize Automation Terraform Provider** (`https://blogs.vmware.com/management/2020/01/getting-started-with-vra-terraform-provider.html`)
- **Network Automation with Cloud Assembly and NSX – four part series** (`https://blogs.vmware.com/management/2019/04/network-automation-cloud-assembly-and-nsx-part-1.html`)
- **Ansible integrations with vRealize Automation – Cloud Assembly:** `https://blogs.vmware.com/management/2019/05/ansible-integration-cloud-assembly.html`

  `https://blogs.vmware.com/management/2020/02/introducing-ansible-tower-integration-with-vrealize-automation.html`
- **VMware vRealize Blog** (`https://blogs.vmware.com/management/2019/08/vrealize-automation-8-whats-new-overview.html`)
- **vRealize Orchestrator and Action Based Extensibility:** Use extensibility to extend the out-of-the-box vRealize Automation experience. `www.vmware.com/products/vrealize-orchestrator.html`

https://docs.vmware.com/en/VMware-Cloud-Assembly/services/Using-and-Managing/GUID-55847415-5920-47E7-86BD-20CD9EB6BA6B.html

- **Ivan Pepelnjak** (https://blog.ipspace.net/)
- **Kirk Byers** (https://pynet.twb-tech.com/)
- **Network Automation in Network Computing** (www.saltstack.com/solutions/netops/)
- **SearchNetworking in Tech Target, subtopic: Network Automation** (https://searchnetworking.techtarget.com/resources/Network-automation-and-intent-based-networking)
- **Networktocode, Curated Awesome list about Network Automation** (https://github.com/networktocode/awesome-network-automation)
- **IP Engineer David Gee** (https://dave.dev/)
- **The Network Automation Thoughts of Michael Kashin** (https://networkop.co.uk/)
- **Packet Life —The Network Automation thoughts of Jeremy Stretch** (https://packetlife.net/)
- **Packet Pushers** (https://packetpushers.net/blogs/)

Online Courses

You can also take an online class to enrich your understanding of network automation. Here are some we recommend, many of which are free or inexpensive:

- **Coursera:** Networking and Security Architecture with VMware NSX
- **Coursera:** Software-Defined Networking
- **IpSpace:** Building Network Automation Solutions
- **Lynda.com:** Python NAPALM Network Automation
- **Pluralsight:**Automating Networks with Ansible the Right Way
- **Pluralsight:** Deploying Network Configuration Manager and Telemetry Solutions

- **Udemy:** Learn Python Network Programming for Network Engineers
- **Udemy:** Learn NetConf, Yang, SDN Opendaylight Netconf connector
- **VMware Education:** Look under NSX-T, Network Virtualization

Webinars

Webinars are another great resource to keep informed about network automations. Check out the following:

- **VMware Network Automation:** Automate NSX Using vRealize Automation
- **IpSpace Webinars:** Look under Network Automation
- **SDx Central, SDN Webinars** (`www.sdxcentral.com/networking/sdn/sponsored/webinars/`)
- **VMware events and webinars:** (`www.vmware.com/company/events.html`)

Podcast Feeds

Podcasts are another great way you can listen to leading network automation experts. Here are a few to start:

- **Network Automation Nerds Podcast** by Eric Chou
- **NSX Ninjas Podcast**
- **Packet Pushers**
- **Software Gone Wild Podcast** by Ivan Pepelnjak

Videos

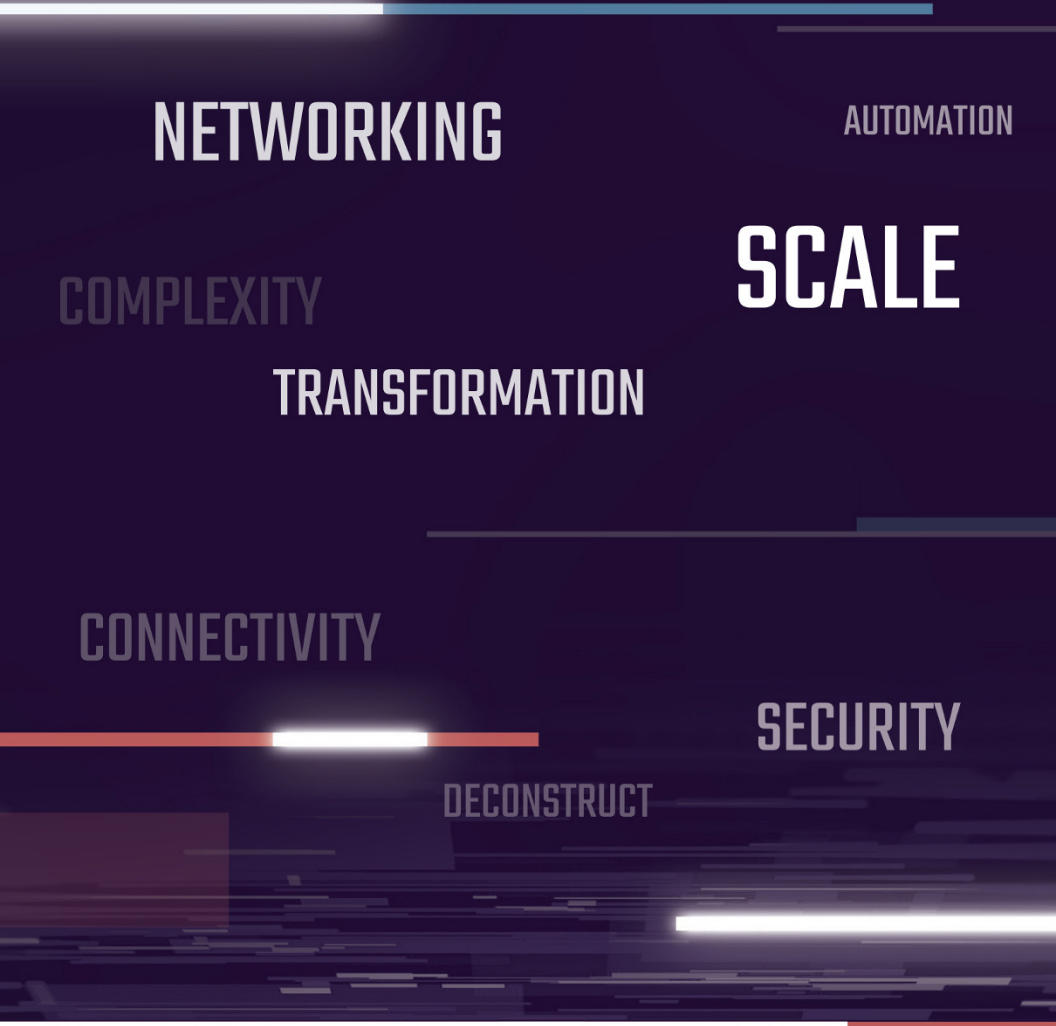
You can find a wealth of videos on network automation from practitioners and trainers. Check out our favorites:

- **David Mahler:** Ansible for Network Configuration Templates
- **INEtraining:** Network Automation with Ansible
- **IPvZero:** Ansible Network Automation with Jinja2
- **IPvZero:** Introduction to Nornir | Python Network Automation!
- **John Anderson:** Network to Code Network Automation Architecture
- **Packet Pushers:** What is the Right Tool for Network Automation? Python? Ansible? Ansible? Salt?
- **Saltstack:** Orchestrating Network Devices with SaltStack – Cloudflare
- **Tech Field Day:** Networking Field Day

Conferences and Meetups

The best way to learn a new technology is to get hands-on experience by going to a conference. Many conferences offer low-cost, pre-conference training workshops, including the following:

- **AnsibleFest** (`www.ansible.com/ansiblefest`)
- **ChefConf** (`www.chefconf.io/`)
- **Future:NET** (`www.vmware.com/futurenet/`)
- **HashiConf** (`https://hashiconf.com/`)
- **Puppetize** (`https://puppet.com/puppetize/`)
- **PyCon** (`https://us.pycon.org/`)
- **SaltConf** (`https://saltconf.com/`)
- **VMworld** (`www.vmworld.com/en/us/index.html`)



FUTURE:NET

Future:NET is the premier networking industry event, led by those who have walked the walk. Engage industry experts and networking leaders as they drive deep technical discussion and offer insight around the future landscape.

Join the conversation.

vmware.com/futurenet

Discover all you need to know about network automation

Digital organizations are moving fast and expect infrastructure changes to happen equally as fast. Networks have to adapt to the rate of change in today's organizations. The only way to keep up with the rate of change with consistency and predictability is to automate. If your organization is looking for how to do that, then *Network Automation For Dummies,* VMware Special Edition, is for you. This handy guide explains everything you need to know about network automation, including what automation is, why automation is important, how you can automate, and so much more. This practical guide also explains in plain English the ins and outs of network automation and how your organization can implement it.

Inside...

- Examine the benefits to network automation
- Get a clearer picture of the automation landscape
- Focus on the ways to automate
- Find out how to incorporate automation in your organization
- Discover additional resources to help you with automation

vmware®

Karl Fultz (@kwfultz) is a Staff Technical Marketing Architect on VMware's Cloud Management Unit. **Madhukar Krishnarao** (github.com/madhukark) is a Technical Product Manager, and **Susan Wu** (@susanwu88) is a Senior Product Marketing Manager. Both Madhu and Susan are in VMware's Networking and Security Business Unit.

ISBN: 978-1-119-69977-4
Not For Resale

for dummies® A Wiley Brand

Also available as an e-book

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.