Compliments of

BROCADE

A Broadcom Limited Company

SAN Automation

for dummies

A Wiley Brand

Rediscover the automated SAN

Modernize your automation toolkit

Learn to automate your first utility

Chip Copper

John Paul Mueller

Brocade Special Edition

About Brocade

Brocade, a Broadcom Limited Company, is the proven leader in Fibre Channel storage networks that serve as the foundation for virtualized, all-flash data centers. Brocade Fibre Channel solutions deliver innovative, high-performance networks that are highly resilient and easier to deploy, manage, and scale for the most demanding environments. The network matters for storage and Brocade Fibre Channel storage networking solutions are the most trusted, widely deployed network infrastructure for enterprise storage.

www.broadcom.com/brocade

SAN Automation

Brocade Special Edition

by Chip Copper and John Paul Mueller

for dummies®
A Wiley Brand

SAN Automation For Dummies®, Brocade Special Edition

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

ISBN 978-1-119-51543-2 (pbk), ISBN 978-1-119-51546-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. Some of the people who helped bring this book to market include the following:

Project Editor: Martin V. Minner
Editorial Manager: Rev Mengle
Executive Editor: Katie Mohr
Business Development Representative: Karen Hattan
Production Editor: Siddique Shaik
Brocade Contributor: Curt Beckmann

Table of Contents

Introduction

The world runs on data. It isn't just businesses any more; it's everywhere, everyone needs it, and it is ever-changing and growing. Accessing all that data requires highly available, lightning-fast access through technologies such as Fibre Channel Storage Area Networks (SANs). With the incredible increase in data storage needs, storage administrators who manage SANs need all the help they can get. That's where automation comes into play. Without automation, trying to get anything done becomes an exercise in frustration and repetition. If you're interested in making your life easier, this book provides precisely what you need in a compact form that won't take you hours to read.

About This Book

SAN Automation for Dummies, Brocade Special Edition, is for storage administrators as well as for automation engineers who may be called upon to collaborate with storage teams to automate storage-related activities.

Storage administrators are busy people. They need to consider tasks such as deploying storage volumes for new virtual machines, updating and verifying software, troubleshooting, watching bandwidth usage, and tracking inventory. This book tells you how automation can help you perform these tasks with greater speed and accuracy than ever before. You also discover how to create your own automation plan so that you create the kind of automation you want with the least effort.

If you currently find yourself using the Command Line Interface (CLI) exclusively, you may want to consider some of the other options discussed in this book. Of course, the discussion begins with why you may want to make a change — it's essential to see a benefit to any change you make. Once you discover there are other methods you can use, you see examples written using the RESTful API approach, PyFOS, and Ansible.

You also aren't left with the idea that this is a fly-by-night solution. This book tells you about the standards-based underpinnings of automation and it helps you discover why this approach works so well.

Icons Used in This Book

As you work through this book, you see various icons meant to attract your attention. This book uses the following icons to emphasize information you may find helpful in particular ways.

REMEMBER

The information marked by this icon is important. If you get nothing else out of the chapter, you need to remember information marked by this icon. This icon also makes it easier to spot noteworthy information when you refer to the book later.

TIP

Everyone likes those special bits of information that make life easier or help perform tasks faster. This icon points out extra-helpful information that you need when creating a SAN automation solution effectively.

WARNING

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

IN THIS CHAPTER

- Exploring the automated SAN
- Comparing to other infrastructures
- Adding Fibre Channel to workflows
- Developing an automation plan

Chapter 1
Rediscovering the Automated SAN

Before we get into the nitty-gritty of automating today's SANs, this chapter looks briefly at how automation has been incorporated in various networking technologies over the years.

The first section of this chapter provides you with three cases where you receive benefits from built-in Fibre Channel (FC) fabric automation without any additional effort on your part.

The next two sections of this chapter tell you a bit about FC history and discuss how you can add FC to your workflow automations. Gaining a good understanding of why FC is different and what it can do for you is important.

In the final section of the chapter, you see how to develop your own automation plan. Without a great plan, any updates you perform to your network would prove disruptive at best.

Exploring the Automated SAN

Although this book discusses automation and FC fabrics, it's important to note that FC has provided some automation for a long time — even decades. The following sections provide you with three examples.

Fabric formation

With many alternative network technologies, there are many steps between taking the switches or routers out of the boxes and configuring them so that they're capable of moving traffic. It takes a lot of planning, and if you get any of the details wrong, your network may not come up, or it may behave unexpectedly.

REMEMBER

FC requires less planning. All you need to do is take the switches and directors out of the box, power them up, and connect them with cables. There's enough automation and intelligence inside of each one of the switches to know how to bring itself up on a fabric and configure itself so that it can communicate with its neighbors. Furthermore, if the network is already handling traffic, the new switch joins in the movement of that traffic without further configuration.

Anyone who's worked with Ethernet networks is familiar with the *spanning tree protocol*, created to deal with bridge loop problems because basic Ethernet cannot form a fabric. IP routers, however, can form fabrics. Many devices support both Ethernet and IP, so you can cause chaos on an Ethernet/IP network if you plug a cable into the wrong port.

FIBRE CHANNEL'S ORIGINS

Although Ethernet was named after the nineteenth-century "luminiferous aether," it's rather younger than that, having been around about four decades now. In the initial shared media days, switches didn't exist, so you couldn't automate them or expect them to create a fabric. As bridges emerged, the Spanning Tree Protocol was created as a quick fix to prevent crippling broadcast storms by disabling redundant paths; however, that also meant that load balancing across redundant paths required special configuration. FC was developed in light of many networking challenges and was designed to solve many of these issues automatically.

REMEMBER

Through built-in automation, FC networks are smart enough to know when devices are configured in a way that could bring down any other network types. FC incurs fewer errors when adding new devices. FC evaluates and uses all traffic routes. There is no risk a new device will bring the fabric down or experience slow reconfigurations because someone plugged the wrong device into the wrong port. In an FC fabric, except for architectural reasons, you can't plug something into the wrong place. Wherever you plug it in, it works.

Trunking

Suppose that you have traffic demands that exceed the capacities of one link between a pair of switches. With other networks, adding a link between an overwhelmed switch pair requires that you perform a complicated set of tasks like this:

1. Enable the ports.
2. Configure the ports, taking care to use exactly the same parameters on both sides.
3. Change parameters where necessary to reflect their cooperative nature.
4. Cable the switches together.

If these steps go well, the links will share traffic, but they may not do so optimally. For example, a single large flow may stick to a particular link. That can mean that one link is barely used while the other link is saturated.

REMEMBER

FC provides an easier solution called *trunking.* All you need to do is to connect pairs of ports in the same port groups across two switches, and the trunk forms automatically. Traffic is automatically load balanced between these two or more links, and you never have the situation where one link is swamped while others remain lightly loaded.

Slow drain device detection

Suppose that you have a storage device on the network that is incapable of accepting storage traffic at a rate at which the host is capable of sending it. This is sort of like a ramp that leads off an Interstate highway but has a stoplight at the bottom. You'll likely see a line form when you get off an exit that has a lot of

traffic, and that line often flows into other lanes so all the traffic on the highway slows down. The same thing happens on a storage network when a slow drain device causes traffic to back up on the SAN.

REMEMBER

The SAN fabric can detect storage traffic backing up to a slow-draining device and automatically moves that traffic to its own separate virtual channel. This feature allows other traffic to continue through the same switch and even on the same links without hindrance from the slow-draining device. This all takes place automatically without any intervention by the network administrator.

Comparing to Other Infrastructures

FC's main early competition was directly attached storage. As external storage arrays grew in capacity, it became clear that, to be cost-effective, they had to be shared among many hosts. Adding ports to storage arrays was expensive so the need for a network was clear. It was soon clear that the storage teams, rather than the networking teams, had to manage these networks because the storage teams understood the unique requirements of storage traffic. *Network-Attached Storage (NAS),* a file-level data storage server, was available, but no one would dream of booting an operating system or putting swap platters on storage connected using Ethernet or TCP/IP.

Over time, the reliability and speed of Ethernet networks improved, so for large data repositories, two competitors emerged — FC and Ethernet. For years, system architects only considered these two alternatives when designing next-generation systems.

Today, new competitors have appeared on the scene. Because of the availability of higher capacity drives, direct attached devices are once again appearing as an alternative to network storage. Other architectural paradigms such as *hyper-converged infrastructure (HCI),* a fully software-defined IT infrastructure that virtualizes all conventional hardware-defined system elements, and cloud services are offering to make storage available to users as well.

The management paradigms used in the early days of network storage were somewhat cumbersome and awkward. Even with FC, you performed a number of managerial steps in a specific order

to allocate a new piece of storage to a host or virtual machine. With the introduction of new storage alternatives, such as HCI and cloud storage, administrators have asked that FC vendors increase the amount of management that's available using application programming interfaces (APIs) and other automation mechanisms. This competition is good for advancing all storage technologies, so now FC integrates into user-based provisioning, allocation, and management mechanisms just as easily as those found with other storage alternatives such as cloud-based storage, direct attached storage, or HCI.

Integrating FC into Today's Workflows

Storage administrators face many tasks they perform on an ongoing basis. Here are some examples of where automation can really help out:

- **Allocating storage for new VMs:** Supporting new virtual machines (VMs) in today's highly virtualized environment is an almost hourly task. Performing this task as efficiently as possible is imperative if the storage administrator is going to keep up with the hectic pace of the virtualization team. Deploying storage to a new VM is an interesting task because it may touch several different platforms using these steps:

  1. Allocate the storage on the storage array.
  2. Present the storage on a particular storage port.
  3. Add the storage port to a SAN zone that contains the names of the servers that will serve the VM.
  4. Configure the server to allow its network adapters to see that particular storage array.
  5. Let the VM know where its storage lives.

  This process has a lot of moving parts. The beauty of using automations is that they can span several platforms. In this case, storage networking automation can ensure that the new storage is allocated to these VMs, and zones are set up to be driven by orchestration systems that may live on the storage array, the host, or on an external orchestration system. The open automation APIs and utilities make it easy to automate the SAN to integrate into whatever provisioning mechanism customers decide to use.

- **Updating and verifying software:** FC fabrics are, by their nature, more secure than their IP counterparts. Nevertheless, the need to ensure constant access to sensitive data requires that SANs be promptly updated with the latest patches and updates. Making sure that every device in the SAN has the latest updates is a high-priority task for network administrators and security officers. SAN automation allows an administrator to set up a schedule for distributing, installing, and verifying new software on all elements in the storage network. Orchestration allows coordination to ensure everything goes smoothly and as expected during updates of other hosts and networking devices.

  In addition, you can automatically run audits to verify proper software installation in the proper places — so that software installers don't encounter any surprises. Automating software installation can also detect and recover from rare but critical software installation failures.
- **Troubleshooting:** If something goes wrong in a storage area network, the first step is to verify SAN configuration. You can automatically run configuration verification automations to guarantee that nothing strays from the approved configuration. If a problem occurs, your automated tools can pull the appropriate diagnostic and analytic information from all fabric elements and make it available to other diagnostic tools to root-cause the problem. Because of the many monitors and counters available in the FC network, the SAN can even help diagnose problems that exist on the host or the storage array. You obtain this information quickly and on demand through the fabric APIs.
- **Inventorying:** Administrators want to know about devices on the storage network. An administrator may want to verify that other locations can still see everything in the storage network, or want something as simple as a listing of all the ports that are occupied or available so that they can plan for the rack placement of the next device. Many system administrators have been surprised to learn on a Monday morning that someone has added a host or storage device to the network without their knowledge. SAN automation tools allow an administrator to gather this information and synthesize it with other information to provide accurate, on-demand snapshots of the storage infrastructure state.

Creating an Automation Plan

Automation isn't magic. Behind every automation is an expert who knows exactly how that automation was created. In some cases, experts work in-house. In other cases, experts come from a partner or a vendor to supply the missing skills and knowledge to create the automation. In any case, it's up to the expert to determine the exact automation scope, the approach to take, the required resources, what constitutes success, how to verify the automation, and how to run the automation.


TIP

An easy guideline to use is that, if you can write a set of steps down in a way that someone who doesn't understand the task can perform them, you can automate the task. If there is any ambiguity in the process or if a certain level of finesse is required to perform an operation in a way that you can't write down or describe, the task may not be a good candidate for automation. Here are some sweet spots for automation solutions:

- **Performing the same tasks repetitively:** There is a good chance you can automate repetitive tasks. For example, suppose that you need to obtain log files every morning and analyze them to be sure that nothing went wrong during the night. In addition, you must remove the log files from the SAN devices to make room for new logs. Even if the log files happen to live on an external device, they still need to be gathered, analyzed, and archived. This well-defined task lends itself well to the idea of automation.
- **Doing something simple on a large number of devices:** You may perform a simple task just once a year, but if the task involves performing the same steps many times, each time with a different set of parameters, you can use automation to perform the task faster and with fewer errors. Consider the case of adding a new storage array to an infrastructure where you want to add the storage array into 50 different host zones.
- **Doing a sensitive thing flawlessly and quickly:** Sometimes you must do something just once, but you know you must get it exactly right, and you only have a small window in which to do it. Automating this activity lets you verify it in a safe context before doing it live.

- **Having experts prepare automation suites that are executed by others:** Every organization has its go-to person whom everyone knows is the in-house resident expert on some particular task. Unfortunately, unlike computers, networks, and storage, people don't scale. When only one person knows how to perform a particular task because of its complexity, then there is a bottleneck in the process. If that person happens to be ill or on vacation when you need the task done, then the task must wait or a non-expert must try to perform it correctly. This problem also works against a self-service model where the users are free to perform tasks that would otherwise require the intervention of IT professionals.

Now that you have a better idea of the use cases that work well for FC, it's time to create an automation plan. The following steps get you started:

1. Find a task that matches one of the sweet spots in the preceding list.
2. Write the steps that are necessary to carry out the task:
   a. Determine how you would perform the task manually.
   b. Find an automation expert who can translate the manual tasks into a series of calls and routines to be made available through the automation APIs and utilities.


TIP

For your first task, choose something that's simple, well understood, and verifiable. Many people choose infrastructure auditing as their first automation task. By only reading and not changing the network, you gain confidence in using automation routines correctly, preparing you to create and deploy more sophisticated automations that will change the storage network configuration in the future.

IN THIS CHAPTER

- Understanding the command line interface (CLI) better
- Relying on structured data exchange
- Considering how to structure data
- Defining the benefits of standardized protocols

Chapter 2
Modernizing Your Automation Toolkit

Administrators and engineers alike rely on the command line interface (CLI) to perform a broad range of tasks — mostly because the CLI is both fast and simple. In addition, scripting CLI can take a variety of forms. The first section of this chapter considers the limitations of CLI.

Even though humans can work quite well with the CLI and its sometimes-convoluted output, computers often can't. The next two sections of the chapter explore the means to ensure good communication using the Network Configuration (NETCONF) protocol standard and methods used to format that data for transmission.

The final section discusses standards that make automation easier. Using standards enhances reliability and makes it possible for independent developers to write output that automatically works with other applications that adhere to the standard.

Considering the Limits of CLI

Most networking engineers are familiar with their SAN device's CLI, which has been around forever. As network operating systems gained features, vendors added graphical user interfaces (GUIs)

with menus and icons. Nevertheless, many network administrators only use a GUI to open a number of console windows at the same time. Thus, vendors have had to add new features to their CLIs as well, by adding new parameters and options to their command syntax.

Unfortunately, this strategy has its problems. From an automation perspective, the biggest problem is that an updated command may need to reformat its output to include new features. Suppose that a CLI command outputs performance counters for a particular port. The updated command adds a new port counter, changing the way it shows the information.


REMEMBER

The good news is that humans are smart. Most network operators can move between network consoles running different code revisions — and even different networking equipment brands — and understand the information as each vendor provides it. They can adjust to changes and interpret what they see on screen. If they don't understand something, they can read a manual or watch a video to explain it, making it possible to understand the command output. Unfortunately, automation can't understand the changes in the CLI output as easily as humans do.

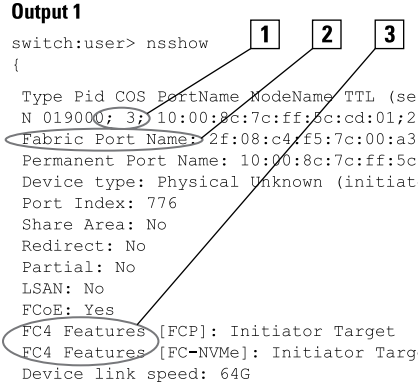
Consider the example shown in Figure 2-1 of the output of two `nsshow` commands. This example was generated by running the same command on two Fibre Channel (FC) switches that differed in their software by only a minor revision number.

The two outputs are similar, and if you can largely understand one of them, you likely can also understand the other. Unfortunately, computer logic looking at these different outputs (from the same command) would encounter several issues. Some are big issues, and some small, but any of them might break the automation. Here are some observations on the differences in output:


REMEMBER

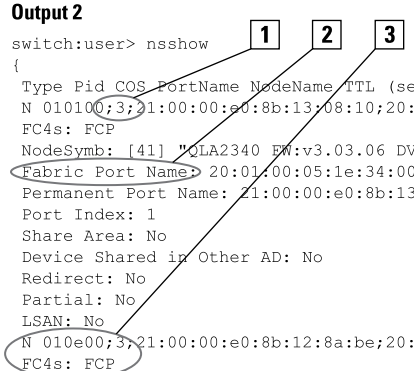
- There is a space after the semicolon in the line that begins with N in Output 2 as indicated by callout 1 in Figure 2-1. That space is missing in Output 2. Surprisingly, this difference may be the one that causes the greatest difficulty. Program logic may expect new line items to appear or disappear, but it is very difficult to anticipate format changes in a previously known line of data output.

Output 1




```
switch:user> nsshow
{
 Type Pid COS PortName NodeName TTL (sec)
 N 019000; 3; 10:00:8c:7c:ff:5c:cd:01;20:00:8c:7c:ff:5c:cd:01; na
 Fabric Port Name: 2f:08:c4:f5:7c:00:a3:20
 Permanent Port Name: 10:00:8c:7c:ff:5c:cd:01
 Device type: Physical Unknown (initiator/target)
 Port Index: 776
 Share Area: No
 Redirect: No
 Partial: No
 LSAN: No
 FCoE: Yes
 FC4 Features [FCP]: Initiator Target
 FC4 Features [FC-NVMe]: Initiator Target Discovery_Service
 Device link speed: 64G
 Connected through AG: Yes
 Real device behind AG: Yes
The Local Name Server has 1 entry }
```

Output 2




```
switch:user> nsshow
{
 Type Pid COS PortName NodeName TTL (sec)
 N 010100;3;21:00:00:e0:8b:13:08:10;20:00:00:e0:8b:13:08:10;na
 FC4s: FCP
 NodeSymb: [41] "QLA2340 FW:v3.03.06 DVR:v9.0.0.2 (w32 IP)"
 Fabric Port Name: 20:01:00:05:1e:34:00:70
 Permanent Port Name: 21:00:00:e0:8b:13:08:10
 Port Index: 1
 Share Area: No
 Device Shared in Other AD: No
 Redirect: No
 Partial: No
 LSAN: No
 N 010e00;3;21:00:00:e0:8b:12:8a:be;20:00:00:e0:8b:12:8a:be;na
 FC4s: FCP
 NodeSymb: [41] "QLA2340 FW:v3.03.06 DVR:v9.0.0.2 (w32 IP)"
 Fabric Port Name: 20:0e:00:05:1e:34:00:70
 Permanent Port Name: 21:00:00:e0:8b:12:8a:be
 Port Index: 14
 Share Area: No
 Device Shared in Other AD: No
 Redirect: No
 Partial: No
 LSAN: No
 Port Properties: SIM Port
The Local Name Server has 2 entries }
```

FIGURE 2-1: Running a command on two different versions of FC software.

- The ordering of some of the elements is different. Fabric Port Name has changed position, as indicated by callout 2 in Figure 2-1.
- Each output contains elements not contained in the other, as shown by callout 3 in Figure 2-1. Output 1 has lines labeled FC4 Features; Output 2 has a line labeled FC4s.

REMEMBER

DEFINING AUTOMATION COMMUNICATIONS

Because of the need for automations to communicate with networking devices, the Internet Engineering Taskforce (IETF), working with industry leaders, developed the Network Configuration (NETCONF) protocol standard. This standard defines a mechanism that allows automations to communicate effectively with network devices to obtain and set state and configuration information along with receiving event notifications from network devices. NETCONF can remove many of the pitfalls associated with attempting to communicate with a network device using the CLI or other non-standard mechanisms.

NETCONF establishes a common model that represents the state and configuration information in a switch. It's the responsibility of the switch or network device operating system to translate values represented in this model into the operational parameters used by the switch. This allows a layer of abstraction between the state that the automation or orchestration tool is trying to create and the actual implementation of that state in the network device.

For years, network administrators have used mechanisms like send/expect scripts to try to parse CLI output, but as networks continue to get more sophisticated, so does their output, and consequently this task becomes more difficult.

Using Structured Data Exchange

The model that is used to represent state and configuration information is expressed in a modeling language called Yang. Yang describes the structure of the different elements inside the model, and is used to describe whether each element is read-only or read-write. It describes the type of data that the element can hold, such as string or integer, and it shows the relationship among various elements, the other nested elements they contain, their peer elements, and the parent elements that contain them. Here is a segment of the description of a zone in Yang:

```
list zone {
  key "zone-name";
  description
    "List of the members in the zone. The members can
     only be identified as a WWN, domain, index, or a
     zone alias.";
  leaf zone-name {
    type zoning-name-type;
    description
      "The zone name.";
  }
  leaf zone-type {
    type zone-type-type;
    description
      "The zone type.
       Not that target zone types cannot be created
       or modified (only deleted).";
  }
  container member-entry {
    description
      "The zone member.";
    leaf-list entry-name {
      type zone-member-type;
      min-elements 1;
      description
        "List of the members in the zone. The members
         can only be identified as a WWN, domain,
         index, or zone alias.";
    }
    leaf-list principal-entry-name {
      when "../..zone-type=1 or ../../zone-type=2";
      type zone-member-type;
      min-elements 1;
      description
        "List of the principal members in the peer
         zone. The members can only be identified as
         a WWN, domain,index, or zone alias.";
    }
  }
}
```

Ordinarily more information goes into a Yang module such as revisioning and governance information; this listing omits them for brevity. Thus, the Yang description is complete, but it's also wordy. Although this precision is necessary when interacting with the model programmatically, it's sometimes useful to get a global view of the abstraction provided by the model to see how the data is structured.


TIP

An open source tool called `pyang` can parse the Yang model and produce a tree that represents the elements in the model. The listing includes information about each element, such as whether it's read-only or read write, a list, optional, or nested. Here is the representation of the zoning model in tree form:

```
+--rw zoning
   +--rw defined-configuration
   |  +--rw cfg* [cfg-name]
   |  |  +--rw cfg-name       zoning-name-type
   |  |  +--rw member-zone
   |  |     +--rw zone-name*   zoning-name-type
   |  +--rw zone* [zone-name]
   |  |  +--rw zone-name       zoning-name-type
   |  |  +--rw zone-type?      zone-type-type
   |  |  +--rw member-entry
   |  |     +--rw entry-name*   zone-member-type
   |  |     +--rw principal-entry-name*
   |  |        zone-member-type
   |  +--rw alias* [alias-name]
   |     +--rw alias-name      zoning-name-type
   |     +--rw member-entry
   |        +--rw alias-entry-name*   union
   +--rw effective-configuration
      +--rw cfg-name?              zoning-name-type
      +--rw checksum?                    string
      +--rw cfg-action?                  uint8
      +--rw default-zone-access?         uint8
      +--ro db-max?                      uint32
      +--ro db-avail?                    uint32
      +--ro db-committed?                uint32
      +--ro db-transaction?              uint32
      +--ro transaction-token?           uint32
      +--ro db-chassis-wide-committed?   uint32
      +--ro enabled-zone* [zone-name]
```

```
+--ro zone-name       zoning-name-type
+--ro zone-type?      zone-type-type
+--ro member-entry
   +--ro entry-name*             union
   +--ro principal-entry-name*   union
```

NETCONF has provided the standard way to represent the information inside each network device, and you can exchange that information by using Yang models, which remove the ambiguity found in command lines. The format of the data as exchanged along with the protocol that is used to exchange that data is found in the following chapters.

Understanding the Data Structuring Methods

Yang provides a description of how configuration and state data should be organized inside the networking devices, but you also need to know how information passes between an automation application and the networking device. The initial approach was to allow networking vendors to decide independently how automations interact with their devices. Although flexible for networking vendors, it is harder to create solutions that support products from different vendors.

A second approach is creating a standardized binary representation of information contained in a Yang model. This technique is efficient from a bandwidth viewpoint, but difficult to debug using common network monitoring tools. Because of the amount of bandwidth generally available in the management plane, that degree of efficiency is arguably unnecessary.

The preferred solution is to create a human-readable data representation that relies on commonly understood and implemented rules. Two great candidates are available to accomplish this: eXtensible Markup Language (XML) and JavaScript Object Notation (JSON). XML has been around a little longer and every modern platform supports XML, and nearly all programming languages already have libraries for manipulating XML. JSON is slightly newer, but is a little more human friendly and very popular among programmers. The level of support for JSON is already close to that of XML, and it is likely to match it soon. This section looks first at XML, and then JSON.

REMEMBER

XML marshals data and shows the relationships between data elements. You can create descriptive tags to indicate what data content between tags represents because XML is extensible. In most cases, humans and automations alike easily understand data exchange content.

The power of XML for NETCONF is that the structure of Yang models map nearly identically into the structure of XML. This provides a natural way to represent Yang model elements using an XML representation for transport across the network. Notice the readability of the following XML data block and that it maps exactly into the description of a zone found in the previous section:

```
<zone>
  <zone-name>Multipath_FlashArray</zone-name>
  <zone-type>0</zone-type>
  <member-entry>
    <entry-name>FlashArray_6112</entry-name>
    <entry-name>BigArray66_5d3d00</entry-name>
  </member-entry>
</zone>
```

JSON is structurally similar to XML. Although the name contains JavaScript, it's a language-independent mechanism for representing and exchanging data that uses fewer characters than XML, making it more bandwidth-efficient and a little easier to edit by hand. The following code shows the JSON form of the XML-based zoning information:

```
{
  "zone": {
    "zone-name": "Multipath_FlashArray",
    "zone-type": "0",
    "member-entry": {
      "entry-name": [
        "FlashArray_6112",
        "BigArray66_5d3d00"
      ]
    }
  }
}
```

Both XML and JSON are found in NETCONF implementations, and some devices allow an incoming request to specify which representation is to be used for the data exchange. Many programmers prefer one representation over the other. Libraries and utility modules are available that will translate one into the other if necessary.

Using Standardized Protocols

NETCONF calls to a network device take on the appearance of remote procedure calls. A program creates a message bundling an operation along with any related data. That message is sent to a network device where the device takes any necessary action, gathers necessary data, and returns the result.


TIP

This process should sound familiar. This is what you do every time you use a web browser. You type in the URL, which often includes some parameters, and then the browser sends a GET request to the remote host. The remote host parses your message, performs necessary operations, and returns the results, usually in the form of an HTML document that may have attachments.

For these reasons, many NETCONF implementations use HTTP to transport messages to and from network devices. A big advantage of HTTP is that, because so many platforms implement it, HTTP is well understood, and interoperability between different clients and servers is almost guaranteed. In addition, the contents of HTTP messages are readable by humans.

To see this feature in action, the following code shows a trace of the first part of the HTTP transaction used in the previous sections. (For brevity, it shows only the meaty bit— not the early bit where the client logs into the switch, nor the later bit where the well-behaved client logs out.) Here is the request for device configuration:

```
GET /rest/running/zoning/defined-configuration HTTP/1.1
Host: 10.18.254.37
Accept-Encoding: identity
Content-Length: 0
Authorization: Custom_Basic
YWRtaW46eHh4OmYzNzA3MGYzM2VhMDI5ZDR5MTZiNjU0ZGE1N2E4O
TU5OTV1ZjRjNzU2ZTk3NmU0ZGEzM2U3ZTN1YWQ1NDM3Yjk=
```

An excerpt from the switch response shows the results:

```
HTTP/1.1 200 OK
Date: Mon, 29 Jan 2018 20:31:12 GMT
Server: Apache
Cache-Control: no-cache
X-Frame-Options: DENY
Content-Secure-Policy: default-src 'self'
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Connection: close
Transfer-Encoding: chunked
Content-Type: application/yang-data+xml

2000
<?xml version="1.0"?>
<Response>
<defined-configuration>
<cfg>
<cfg-name>CFG_FABRIC_A</cfg-name>
<member-zone>
...
<zone>
<zone-name>Multipath_FlashArray</zone-name>
...
</zone>
...
</cfg>
...
</defined-configuration>
</Response>
```

If you look inside an HTTP request in your browser, you may notice that something looks a little different here. Although the preceding NETCONF message uses HTTP, the message contents are not HTML, but XML (or, in similar cases, JSON).

IN THIS CHAPTER

- Seeing the solution options
- Making a decision

Chapter 3
Choosing Your Approach

In this chapter, you discover the various automation options that are available, consider the advantages of each, and see some sample code showing each approach. Of course, the approach you choose is based on all sorts of issues that this book can't consider, such as availability of developers with the proper skills. At the end of the chapter, you consider how to choose an approach and move forward.

Considering the Available Choices

Before we get into the details of any specific implementation methodology, you have a few things to think about. Looking at the different automation approaches without having a project in mind is like going to a store and browsing for tools without knowing how they work or what you want to build. Although you might learn something, if you don't have a project in mind, it's doubtful that you'll be ready to automate something that has value to you or your company.


TIP

You should practice with different automation types before you dig into your first serious project, but it's important to keep your goals in mind. Your initial automation experiments should familiarize you with the toolset that you want to use for your final project. Otherwise, you spend too much time learning

programming languages, command line tools, or markup languages that you won't use. Before reading the following sections, take a few moments to think about your environment and the tasks you do every day that you could automate. For example, you might run diagnostic queries to determine the cause of common problems in your infrastructure. You might also automate answers to frequently asked questions.

Pretend you have a robot assistant to perform this task for you. Think about the training steps used to train a robot that knows nothing about your environment to perform the task successfully. You use these steps to capture the essence of the task so that you can automate it. Now that you have the steps in mind, it's the time to consider your approach:

- **Nuts and bolts:** You may need to perform the task at the nuts-and-bolts level. There are intricate tasks that you carefully and perform in a unique way because of your particular situation.
- **Integration:** You might not limit the motivation for the automation to the scope of the storage network. Your final task may include putting the information into an easily consumed format for users who don't know the details of your infrastructure. That means integrating toolsets and modules that go beyond the boundaries of the SAN.
- **Declarative:** Your mindset may be that you don't want to have to think about each little step that goes into network configuration. The ability to express what the network is supposed to look like and have the infrastructure automate itself by migrating it from where it is to where you want it to be is important.

The good news is that each of these cases has an approach that suits it best. This isn't a one-size-fits-all scenario. The methodology used for the automation depends on the nature of the problem. You could probably use a different toolset for accomplishing the same task, but it would be like building a doghouse using only a screwdriver. You could do it, but it wouldn't be easy.

Using the RESTful Approach

This section uses command line experiments to demonstrate the RESTful approach. You won't implement a large-scale automation using the techniques found here, but the command line approach helps you understand how RESTful interaction happens. Ordinarily, this is done programmatically. In Chapter 4, a Python program handles the required interactions.

This example uses the `curl` command. `curl` is a CLI tool that can transfer data to and from a server using a variety of protocols. For clarity, this example uses `curl` to perform HTTP transactions.

REMEMBER

To interact with a SAN (or other) device, you need to consult its RESTful API reference to learn, among other things, what "Uniform Resource Identifiers" (URIs) you need to use. (Simply put, URIs are identifiers that can be used as part of a web address.) According to the documentation, the URI for accessing a listing of the zones in the active configuration is as follows:

```
GET <base_URI>/running/zoning/defined-configuration/
```

In this example, the `<base_URI>` is `http://<our device IP address>/rest`. Begin by creating a login session with a switch in the fabric by executing the following command (which you'd type as a single line):

```
curl -X POST -v -u admin:password
http://10.18.254.37/rest/login
```

UNDERSTANDING RESTFUL

The RESTful API approach lets you think of a network device as a web server. By using standard web-based tools, an automation can send and receive transactions to or from a network device just as it would send transactions to and from a website. This means that transactions take place over a secure socket using HTTP rules to handle the exchange. The data appears in the form of XML or JSON depending on the RESTful API services implemented on the networking device.

These are the elements that make up the command:

- `curl` is the name of the command.
- `–X POST` specifies the POST HTTP method (instead of GET).
- `–v` specifies verbose output to access the authorization string in the header of the response used in the next step.
- `–u admin:password` specifies the credentials to use.
- The last parameter is the Uniform Resource Identifier (URI) for `curl` to use to login. (URI value is described in the RESTful API reference.)

Here is a trace of the transaction:

```
*   Trying 10.18.254.37...
* Connected to 10.18.254.37 (10.18.254.37) port 80
  (#0)
* Server auth using Basic with user 'admin'
> POST /rest/login HTTP/1.1
> Host: 10.18.254.37
> Authorization: Basic YWRtaW46cGFzc3dvcmQ=
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Wed, 31 Jan 2018 16:01:24 GMT
< Server: Apache
< Authorization: Custom_Basic
YWRtaW46eHh4OjNkYTllZmM3NzMxYjk4OGU2ODg1YzZkMGRjNWJlM
zMyNjBhZDYxZThkOWQ2MWMxNzNiMGV1MjU3YmM2OTcyYjA=
< Cache-Control: no-cache
< X-Frame-Options: DENY
< Content-Secure-Policy: default-src 'self'
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: application/yang-data+xml
```

This command establishes the session used for the following commands. Next, you perform a GET of the URI to return the current configuration.

```
curl -v -H "Authorization: Custom_Basic
YWRtaW46eHh4OjNkYTllZmM3NzMxYjk4OGU2ODg1YzZkMGRjNWJlM
zMyNjBhZDYxZThkOWQ2MWMxNzNiMGVlMjU3YmM2OTcyYjA="
http://10.18.254.37/rest/running/zoning/defined-
configuration
```

By default, the `curl` uses the GET method, so you don't need to specify it. `-H "Authorization: Custom_Basic YWR...jA="` is the authentication and session identifying string returned in the previous command. `-H` places the string into the GET request header as seen in the following trace:

```
*   Trying 10.18.254.37...
* Connected to 10.18.254.37 (10.18.254.37) port 80
  (#0)
> GET /rest/running/zoning/defined-configuration
  HTTP/1.1
> Host: 10.18.254.37
> User-Agent: curl/7.47.0
> Accept: */*
> Authorization: Custom_Basic
YWRtaW46eHh4OjNkYTllZmM3NzMxYjk4OGU2ODg1YzZkMGRjNWJlM
zMyNjBhZDYxZThkOWQ2MWMxNzNiMGVlMjU3YmM2OTcyYjA=
>
< HTTP/1.1 200 OK
< Date: Wed, 31 Jan 2018 16:09:39 GMT
< Server: Apache
< Cache-Control: no-cache
< X-Frame-Options: DENY
< Content-Secure-Policy: default-src 'self'
< X-Content-Type-Options: nosniff
< X-XSS-Protection: 1; mode=block
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: application/yang-data+xml
<
<?xml version="1.0"?>
<Response>
<defined-configuration>
<cfg>
<cfg-name>CFG_FABRIC_A</cfg-name>
<member-zone>
<zone-name>CLUSTER1</zone-name>
```

```
<zone-name>Z_AIXHOST_FCS2_VMAX01_SN1234_9F0
</zone-name>
...

<alias>
<alias-name>esx66_5d3d00</alias-name>
<member-entry>
<alias-entry-name>10:00:8c:7c:ff:5d:3d:00
</alias-entry-name>
</member-entry>
</alias>
</defined-configuration>
</Response>
* Closing connection 0
```

The results appear as an XML data segment structured according to the description in the Yang model, and so it's important to have access to that model along with the RESTful API manual. Having retrieved this information, you should close the session using the CLI command (the results are omitted to save space):

```
curl -v -H "Authorization: Custom_Basic
YWRtaW46eHh4OjNkYTllZmM3NzMxYjk4OGU2ODg1YzZkMGRjNWJlM
zMyNjBhZDYxZThkOWQ2MWMxNzNiMGVlMjU3YmM2OTcyYjA="
http://10.18.254.37/rest/logout
```

As you can see, using RESTful APIs and HTTP exposes you to lots of little details. At times that's exactly what you need. At other times, it's more burdensome than helpful.

Using the Python Approach

One advantage of a higher-level language like Python is that it lets you take a logical view of a task, rather than concentrating on nitty-gritty details. Modules and support functions take a macroscopic view of tasks. A programmer doesn't usually need to worry about local variables, pointers, and data elements that don't relate directly to the solution.


REMEMBER

Open source libraries like PyFOS provide a great starting point for implementing automations on a SAN fabric that supports a RESTful API. For example, they provide insights on implementing fabric RESTful API calls directly. In the previous section, `curl`

communicated with a SAN device. The first step was to log in to the network. The PyFOS library file `pyfos_login.py` shows how implement a login programmatically in Python:

```
import http.client as httplib
LOGIN_RESTCONF = "/rest/login"
...
def login(user, password, ip_addr, isHttps):
    if isHttps == "1":
        conn = httplib.HTTPSConnection(ip_addr)
    else:
        conn = httplib.HTTPConnection(ip_addr)

    auth = user + ":" + password
    auth_encoded = base64.b64encode(auth.encode())

    credential = {"Authorization": "Basic " +
                  auth_encoded.decode(),
                  "User-Agent": "Rest-Conf"}
    conn.request("POST", LOGIN_RESTCONF, "",
                 credential)

    resp = conn.getresponse()
    auth = resp.getheader('authorization')

    if auth is None:
        errors = pyfos_util.set_response_parse(resp)
        if 'errors' in errors:
            return {"login-error": errors['errors']
                    ['error']['error-message']}
        elif 'server-error-message' in errors:
            return {"login-error":
                    errors['server-error-message']}
        else:
            return {"login-error":
                    "unknown login error"}
    else:
        return {"Authorization": auth}
```

This code segment provides valuable insights. It shows:

- The `httplib` module is useful to communicate directly with the RESTful API.

- Module usage for use in other programs.
- How to capture the authorization string used in other operations.
- How to detect login errors.


TIP

You can also use the library routines to forget about the details when they provide a needed higher-level function. The `login()` function avoids most of the messy details of setting up the HTTP connection. Including this module in a Python program allows it to log in to the fabric using the `login()` function:

```
session = pyfos_auth.login(inputs["login"],
                           inputs["password"],
                           inputs["ipaddr"], isHttps)
if pyfos_auth.is_failed_login(session):
        print("login failed because",
              session.get(pyfos_auth.CREDENTIAL_KEY)
              [pyfos_auth.LOGIN_ERROR_KEY])
        usage()
        sys.exit()
```

The associated error recovery actions in the preceding code are from another PyFOS library module. Utilities in the PyFOS repository do the work of assembling low-level routines such as logging into and out of the network into functions that combine many RESTful API calls into a higher level abstraction. In the previous section, three commands obtained current fabric configuration information. The user was required to cut the proper header information out of the login response and use it to build the following two commands.

Even though a program could repeat these three steps, it isn't necessary because a PyFOS library utility has this feature. The `cfgshow` utility accepts an IP address and login credentials from the command line and performs all three operations. The results appear on screen. The only unusual flag is `-f`, which indicates the target virtual fabric. This example doesn't use virtual fabrics, as indicated by the `-1` value.

```
cfgshow.py -i 10.18.254.37 -L admin -P password -f -1

{
    "defined-configuration": {
```

```
        "alias": [
            {
                "alias-name": "AIXHOST_FCS0",
                "member-entry": {
                    "alias-entry-name": [
                        "10:00:00:00:c9:c6:1d:56"
                    ]
                }
            },
            {
                "alias-name": "AIXHOST_FCS2",
                "member-entry": {
                    "alias-entry-name": [
                        "10:00:00:00:c9:c6:12:b2"
                    ]
                }
            },
...
            {
                "member-entry": {
                    "entry-name": [
                        "10:00:8c:7c:ff:ae:92:00",
                        "20:01:00:11:0d:39:01:00"
                    ]
                },
                "zone-name": "host180_9200",
                "zone-type": "0"
            },
```

THE PYTHON 3 DIFFERENCE

Python currently has two completely separate versions available: Python 2.*x* and Python 3.*x*. With most languages, an older version gives way to a newer version. This isn't the case with Python because Python 2.*x* enjoys such a huge advantage over Python 3.*x* in terms of library support for disciplines such as data science. Until recently, a data scientist wouldn't even consider moving to Python 3.*x* because it wouldn't be possible to perform certain tasks.

(continued)

(continued)

However, Python 3.*x* is essential because it fixes many problems in Python 2.*x*, such as inconsistencies in the implementation of certain features. For example, in Python 2.x it's perfectly legal to use either `print "Hello"` (statement form) or `print("Hello")` (function form), which proves confusing. Python 3.x gets rid of the confusion by making only the function form legal.

The `print()` function issue is immediately noticeable because Python issues a syntax error if you use the wrong form. Not so noticeable, but troublesome, is that integer math works differently in Python 2.*x* and Python 3.*x*. The expression `3 / 2` outputs a value of `1` in Python 2.*x*, but a value of `1.5` in Python 3.*x*. In this case, the code executes without a syntax error in either environment, so the issue can go unnoticed. *Beginning Programming with Python For Dummies,* 2nd Edition, by John Paul Mueller (Wiley), can help you get up to speed with Python 3.*x* quite quickly.

REMEMBER

You can call this module from within another program and use the output as input for a following automation stage. Note that this output is in JSON form rather than XML. JSON is easier to use in Python programs, though XML can also be supported.

Using the Ansible Approach

The previous two sections show examples of approaches that use a procedural methodology. The workflow starts at the beginning, executes a series of steps, runs to completion, and then terminates. Most traditional programs work this way.

REMEMBER

Ansible takes a declarative approach. Rather than provide sequential steps, Ansible describes each of the hosts in an inventory. The description appears in a document called a *playbook*. For example, rather than provide steps to install a particular application, Ansible describes a host state where the application is already installed. When you run the playbook, Ansible takes no action if the application is already installed. If the application isn't installed, Ansible calls installation routines so the host is brought into the desired state without requiring the administrator to write any specific steps.

In the realm of storage networks, the use of a declarative language means you can describe switches and fabrics where, for

example, a zone is already configured with the proper hosts and storage arrays. When you run the Ansible playbook, those zones are defined as needed, and the hosts and storage arrays are added to them if necessary.

With some other declarative automation utilities, it's necessary to install an agent on each host that the utility manages. This agent retrieves the commands from a command center and runs them on the local host. Ansible is unique in that it doesn't require agents. In order to make host state changes, Ansible establishes a secure shell session and sends a small Python script to the host. The script carries out the necessary operations and removes itself from the host.

REMEMBER

You need two different skill sets to successfully implement an Ansible solution. First, you must understand the most common playbook operations. These operations are coded and installed for use by the playbooks. As vendors announce support for Ansible, they also provide script libraries for the most common tasks. If there is a task that is required but isn't available in the official Ansible distribution, the open source community may provide code for that task in publicly available repositories.

Second, you must understand your business needs to provide ongoing playbook development. The person maintaining the playbooks doesn't need to be a programmer and doesn't need to know how remote system operations occur. That person only needs to know the desired outcomes, and should be able to construct playbooks in YAML — the markup language used by Ansible. This is an example of an Ansible playbook:

```
---
- hosts: fc_switch_1

  vars_files:
    - ../vars/fos_password.yml
  vars:
    port_name: "0/0"

  tasks:
  - name: enable "{{port_name}}"
    m_port_op_enable:
        switch_ip: "{{fos_ip_addr}}"
        user: "{{fos_user}}"
```

```
        password: "{{fos_password}}"
        vfid: -1
    name: "{{port_name}}"
    mode: "True"
```

The three dashes at the beginning are part of the YAML specification. The hosts section identifies automation target switches. You can keep sensitive information in a separate file, as demonstrated by the `fos_password.yml` line. The name of a variable in double braces: `{{port_name}}` indicates variable substitution. The variable file specified in `var_files` tells where to find external variables.

Making Your Decision

In addition to programming style, other circumstances can cause you to choose one programming model over the other. For example, you may know a programming language other than Python. Fortunately, many languages have a library that allows RESTful API calls. All you really need then is an example of how to make the RESTful API call and adapt it for use in your storage network.

Python is easy to understand (it's often used in schools for that reason). Reading the Python library code can be helpful. Even if you don't use the Python utilities in their current form, you can still learn a lot by looking at the way the utility functions perform their tasks. Make note of the sequence of calls, the parameters, and return values, and then mimic them in your script.


TIP

Using an interpreted language is often faster than relying on a compiled language. It can be frustrating to repeatedly compile and debug the program, and generally speaking, the toolchain is more complicated for compiled programs. Find a programming language that allows interactive development. Before you commit anything to a script, try doing it interactively to see if it will work.

You may have an automation team already in-house. If this is the case and you will direct the automation development, rather than writing it, you may want to find which libraries the team used in the past, and find someone experienced in web-based programming. Remember that although that person is good at automation, the individual may not be good at storage networking. Take the time to document your procedures before handing them over to the automation expert. You may even consider having the expert do a dry run with the command line interface before he or she tries to automate.

IN THIS CHAPTER

- Understanding the examples
- Working with RESTful applications
- Working with PyFOS applications
- Working with Ansible applications
- Obtaining assistance

Chapter 4
Automating Your First Utility

Nothing helps make a technology feel more real than examples showing how to use it, which is the purpose of this chapter. You see three example types: RESTful, PyFOS, and Ansible. Each example shows a different approach to handling automation.

Seeing the examples is almost certainly going to raise some questions in your mind. You'll probably have even more questions as you move on to create your own automation examples. The final section of this chapter tells you what sort of help is available and where to get it.

Considering the Examples

This chapter presents three approaches to implementing SAN automations. You may find that one of the three approaches suits your particular situation well, or you may decide that a hybrid mix would be better. There is no one-size-fits-all solution.

WARNING

The examples in this section have been kept short and simple for readability. That means that you won't see many safeguards that a production setup would ordinarily need. (Fear not; you can find pointers to production-ready examples later in this chapter under "Getting Help.") Addresses and names are hardcoded into the applications, no error checking is done for the return codes of functions, and there is no exception handling. This approach is used to make the examples easily understood. Each approach has its own set of best practices, and you should follow these best practices carefully.

If, after reading these examples, you decide to implement your first automation, you may consider starting with something that is read-only. This approach presents the lowest risk to the infrastructure, but still gives you the opportunity to work with the tools and the data formats used when creating automations that both read and write information.

TIP

You may also want to take advantage of switches before they go into production. In one case, the first automation project for an enterprise was to pre-provision devices the enterprise would eventually roll out to the network. The project allowed the network engineers to gain experience with automation before the systems went live. The engineers could manually correct any mistakes made during this project before the devices' deployment.

Developing a RESTful API Example

Chapter 3 contains an example of how to access the RESTful API of a fabric using the command line. By contrast, the next example shows how to perform the task programmatically. The goal is to build a simple Python 2.7.*x* utility that takes the IP address and credentials of a switch and displays the firmware version installed on that switch. This example requires no software other than the standard Python installation and commonly available modules. Here is the code that carries out this task:

```
#!/usr/bin/env python


import requests
import xmltodict
```

```
BASE = "http://10.18.254.37/rest"

# Log into the switch and get our authorization code
switchReply = requests.post(BASE + "/login",
 auth=("admin","password"))
requestHeaders = { 'authorization':
  switchReply.headers['authorization'] }

# Get the firmware version running on the switch and
# print the XML results
queryResults = \
  requests.get(BASE +
    "/running/switch/fibrechannel-switch/",
  headers=requestHeaders)
print queryResults.text

# Convert the XML to a Python dictionary and print
# the firmware version
dictionaryResults =
  xmltodict.parse(queryResults.text)
print "Firmware version: " + \
  dictionaryResults['Response']
    ['fibrechannel-switch']['firmware-version']

# Free the session on the switch and verify that the
# logout executed successfully
finalReply = requests.post(BASE + "/logout",
  headers=requestHeaders)
print "Final response code: " +
      str(finalReply.status_code)
```

Because the values for the IP address, username, and password are hardcoded into the application, you can execute this at the command line with no parameters:

```
$ ./example1.py
```

Executing this command produces the following results:

```
<?xml version="1.0"?>
<Response>
  <fibrechannel-switch>
```

```
    <name>10:00:c4:f5:7c:d3:c3:ae</name>
    <domain-id>1</domain-id>
    <fcid>16776193</fcid>
    <user-friendly-name>NewName</user-friendly-name>
    <enabled-state>2</enabled-state>
    <up-time>1284325</up-time>
    <domain-name>englab.brocade.com</domain-name>
    <principal>1</principal>
    <ip-address>
      <ip-address>10.18.254.37</ip-address>
    </ip-address>
    <model>170.0</model>
    <firmware-version>v8.2.0</firmware-version>
    <vf-id>-1</vf-id>
    <fabric-user-friendly-name>TestFabric_AV
    </fabric-user-friendly-name>
    <ag-mode>1</ag-mode>
  </fibrechannel-switch>
</Response>

Firmware version: v8.2.0
Final response code: 204
```

The output begins with a dump of the XML that was returned from the switch. Although it's not required, it's useful to see what was returned as a result of the RESTful API call. Using standard library routines, the XML is parsed and the firmware version information is extracted and printed. The final response code is printed to verify that the session was shut down on the switch.

Developing a PyFOS Example

The PyFOS library located on GitHub (`https://github.com/brocade/pyfos`) gives you the ability to rely on an abstraction layer between the RESTful API calls required for network communications and the logical functionality that takes place after establishing contact. In this example, the PyFOS libraries change the name of a switch as you might do during the initial deployment of that switch. This example relies on Python 3.*x* (see the sidebar "THE PYTHON 3 DIFFERENCE" in Chapter 3 for more information on how Python 3.*x* is different).

```
#!/usr/bin/env python3

import pyfos.pyfos_auth as pyfos_auth
import pyfos.pyfos_switch as pyfos_switch
import pyfos.pyfos_util as pyfos_util
import sys
import brcd_util

USEHTTPS = "0"
USERNAME = "admin"
PASSWORD = "password"
IPADDRESS = "10.18.254.37"
NEWNAME = "NewName"
VFID = -1

# Establish a session with the switch.
session = pyfos_auth.login(USERNAME,
  PASSWORD,IPADDRESS, USEHTTPS)

# Specify the virtual fabric. -1 indicates no virtual
# fabric is selected.
pyfos_auth.vfid_set(session, VFID)

# The name of the switch (the WWN) is required to
# update the switch information.
# Obtain the current name of the switch and put it
# into our reply data structure.
current_switch =
  pyfos_switch.fibrechannel_switch.get(session)
switch = pyfos_switch.fibrechannel_switch()
name = current_switch.peek_name()
switch.set_name(name)

# Set the human-friendly name of the switch
switch.set_user_friendly_name(NEWNAME)

# Send the request to the switch and print the
# result.
result = switch.patch(session)
pyfos_util.response_print(result)

# Release the session
pyfos_auth.logout(session)
```

Prior to the execution of this program, the name of the switch was `OldName` as can be seen in this command prompt:

```
$ ssh admin@switch1
OldName:admin> exit
```

When the program is run, the results are printed in JSON:

```
$ ./example2.py
{
    "success-code": 204,
    "success-message": "No Content",
    "success-type": "Success"
}
$
```

Logging into the switch shows that the operation did indeed run successfully:

```
$ ssh admin@switch1
NewName:admin> exit
```

Developing an Ansible Example

This example uses Ansible to discover the available ports on a switch. At first it may seem that you need to set up a lot of configuration information across several files to describe this simple action. Two of the features of Ansible are its scalability and its ability to describe what you want done without spelling it out in any programming language (instead, you write a code-like description). Although this example runs a simple operation against one switch, you can easily scale this to enterprise size and describe many tasks without knowing a programming language.


TIP

A storage professional may notice that the words "target" and "host" are being used in the Ansible sections of this book, but don't be confused by the reuse of these words. In a SAN context, the host is typically the computer that is running applications and the target is the storage that is servicing that host. When Ansible is being discussed, host and target refer to the object that is being managed by Ansible. This means that an applications host, and a storage array, and even SAN switches can all be targets of Ansible-driven automation, and the storage processor can be the host upon which an Ansible animation acts.

REMEMBER

The task itself appears in a `.YAML` file called a *playbook.* YAML is considered a "human readable data serialization language." (Originally it stood for "Yet Another Markup Language," but now recursively stands for "YAML Ain't Markup Language," perhaps to suggest it is more readable than most markup languages.) Here is a list of descriptive items you create for this example:

- `hosts`: Indicates an inventory of the target devices for this automation.
- `vars`: Allows variable substitution in the task section. Although the variables are listed here in this case, they could also be provided in an external file that can be maintained separately. This allows a degree of separation between the tasks and a standard list of hosts.
- `name`: Optional; creates a more readable task name (otherwise, the module name is used).
- `tasks`: Describes two separate actions to perform:
  1. Run a module named `m_display_port_availability`. The text following the colon (`:`) on the `name` line is a comment for readability. Ansible includes a number of standard modules, but in this case, you need a SAN-specific module. The variable name substitutions get passed to the module as arguments. Think of `register` as specifying that the temporary variable result will hold the output of this task for a later task.
  2. Run the `debug` module using the `result` of the previous step. This is what produces the program output on the console when you run the playbook. It uses the `register` variable `result` populated in the previous task.

With all these requirements in mind, the following code shows how to create an Ansible description.

```
---
- hosts: switches
  gather_facts: False

  vars:
    fos_ip_addr: "10.18.254.37"
    fos_password: "password"
    fos_user: "admin"
```

```
  tasks:
  - name: display all available ports
    m_display_port_availability:
        switch_ip: "{{fos_ip_addr}}"
        user: "{{fos_user}}"
        password: "{{fos_password}}"
        vfid: -1
    register: result
  - debug: var=result
```

The hosts file contains an inventory of the different hosts that Ansible will help manage. The structure of this file allows you to define logical host names and a number of hosts grouped into categories. In the preceding task, the group of devices in the `[switches]` collection is the automation target.

```
switch1=10.18.254.37

[switches]
switch1 ansible_connection=local fos_ip_addr=10.18.254.37
```


TIP

You find the `m_display_port_availability` task defined in the `m_display_port_availability.py` library file. That's right — the module is written in Python 3.*x*. You've probably realized by now that if you are going to write automation utilities, you'll need to make friends with Python. The first few lines of `m_display_port_availability.py` are revealing:

```
#!/usr/bin/env python3

import pyfos.pyfos_auth as pyfos_auth
import pyfos.pyfos_switchfcport as pyfos_switchfcport
import pyfos.pyfos_util as pyfos_util
...
```

This module takes advantage of the PyFOS library to handle communications between the Ansible runtime utilities and the SAN. Typically, you rely on automation engineers to develop a library of generalized and low maintenance utilities like this. The network administrators then run day-to-day operations using the predefined modules. The command to execute this task is pretty simple:

```
ansible-playbook port_available.yml
```

In this command, `port_available.yml` is the file. Here are the results:

```
PLAY [switches] ************************************
*************************

TASK [display all available ports] *****************
*************************
ok: [switch1]

TASK [debug] ***************************************
*************************
ok: [switch1] => {
    "result": {
        "available_ports": [
            {
                "name": "0/0",
                "port-type": "G_PORT"
            },
            {
                "name": "0/1",
                "port-type": "U_PORT"
            },

  ...

            {
                "name": "0/23",
                "port-type": "U_PORT"
            }
        ],
        "changed": false,
        "failed": false
    }
}

PLAY RECAP *****************************************
*************************
switch1                    : ok=2    changed=0
    unreachable=0    failed=0
```

The PLAY line tells that the process is kicking off and that the group of devices named *switches* will be the target. The next few lines tell the results of each task. The description of the first task is the human-readable string specified on the name line. Because the second task didn't specify an additional name, the output shows the name of the module debug.

Under each task is a status line saying the tasks completed successfully. The output under the second task is the output of that task as produced by the debug module. The result string contains the output of the previous step, which is a list of the available ports on the fabric. In more complex automations, later steps would consume this result, which is why it appears in JSON.

Finally, the PLAY RECAP says that two tasks were run successfully on the group switches. The output states that no steps indicate the device states have changed, it's possible to contact all devices in the switches list, and that no tasks indicate failure.

Getting Help

Everyone needs help from time to time. Good news! You can get help and it won't cost anything. Many have traveled before you down the road to automation and have made their tools and experiences available to you. Here are some sites that may help you along your journey:

- **Python (**www.python.org**):** This easy-to-learn language supports the most sophisticated applications and is being taught in grade schools all around the world.
- **Ansible (**www.ansible.com**):** Leave programming to the programmers and concentrate on the managerial perspectives of your automations through this robust, extensible toolset.
- **Libraries and Utilities (**www.github.com/brocade**):** The PyFOS library, Ansible playbooks, and additional programming information can help you implement your ongoing SAN automation projects.
- **Automation Forums (**http://my.brocade.com**):** Share experiences, get advice, and learn from the community in the Automation Forum. Interact with those who are developing the tools and have used them to solve their business problems.

IN THIS CHAPTER

» Ten ways to make your SAN automation experience better

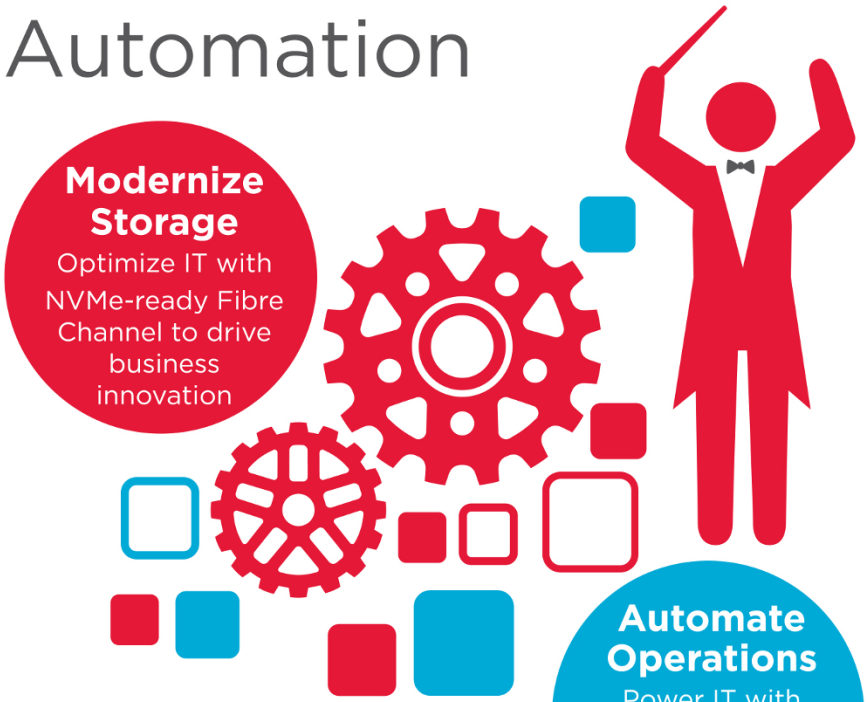
Chapter 5
Ten Ways to Use SAN Automation

Everyone loves helpful tips that make using a new technology easier, which is the purpose of this chapter. Here are some sample cases where building an automation can save time and make your life better:

- **Verifying your fabric:** The operations team tells you that it moved a host from one port to another. A quick automated verification audit could validate the claim.
- **Zoning a new device:** The new storage array arrives and it's time to set up zones for all the hypervisor servers that will use it — all 28 of them! That sounds tedious, but an automation won't mind doing that for you.
- **Provisioning ports to the new server:** You need to generate a list of available ports in every rack at a moment's notice. When the team needs ports for the new host, find them quickly and confidently.
- **Deploying new infrastructure:** It's time to move the switches out of the warehouse and into the data center. Before you deploy them, configure the fabric ID, IP address, and other particulars to make for a smooth, controlled installation.

- **Troubleshooting the fabric:** Every SAN administrator has a go-to set of commands to use as soon as there is a hint of trouble. Some of these commands involve logging into each of the fabric switches to gather initial diagnostic information. Even the first line administrator could run this diagnostic script and hand off the result to the expert to reduce time to resolution.
- **Simplifying complex operations:** Because a script can automate the actions of an expert, you can encapsulate the subtleties of a sophisticated operation so a person with a lower level of expertise can perform them, resulting in more of a self-serve model.
- **Translating reports into another format:** The business units ask for usage reports they can read. You can use scripts to translate the fabric statistics into verbiage the business units understand, and you can generate the reports on the fly. Give the clients current numbers rather than a stale monthly report. You might even let them serve themselves!
- **Repeating timed operations:** Generate that monthly report automatically and have the automation mail it to you. You'll never forget to generate it, and your boss will complement you on your ability to get reports in on time.
- **Integrating with automations on other platforms:** You don't have to limit automations to the SAN. Write a script that gathers information from the fabric and catalogs it into a folder structure on your management server.
- **Chatting with your fabric:** ChatOps allows your infrastructure to participate in conversations with you and teams. When you get that call at midnight, chat with the fabric to ask how it's doing. Get the critical indicators that tell you if things can wait or if you should start your late-night journey to the data center.

Brocade Modernizes Fibre Channel with NVMe and Automation



Accelerate business operations, respond to dynamic demands, and eliminate complexity

Learn More
go.broadcom.com/fc-networking

BROCADE
A Broadcom Limited Company

Automate for speed and accuracy

Storage administrators are busy people. They need to consider tasks such as deploying storage volumes for new virtual machines, updating and verifying software, troubleshooting, watching bandwidth usage and tracking inventory. This book tells you how automation can help you perform these tasks with greater speed and accuracy than ever before. You also discover how to build your own automation plan so that you create the kind of automation you want with the least effort.

Inside...

- Add Fibre Channel to workflows
- Develop an automation plan
- Apply standardized protocols
- Use RESTful, Python, or Ansible
- Explore automation examples

BROCADE
A Broadcom Limited Company

Chip Copper, PhD, is an R&D Principal Engineer at Broadcom. **John Paul Mueller** is a veteran *For Dummies* author.

Go to Dummies.com®
for videos, step-by-step photos, how-to articles, or to shop!

for dummies®
A Wiley Brand

Also available as an e-book

ISBN: 978-1-119-51543-2
Not for resale

WILEY END USER LICENSE AGREEMENT

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