Automating and Orchestrating Networks with NetDevOps

IVO PINTO, CCIE® NO. 57162
FAISAL CHAUDHRY, CCIE® NO. 2706

ciscopress.com

Automating and Orchestrating Networks with NetDevOps

Ivo Pinto, CCIE No. 57162

Faisal Chaudhry, CCIE No. 2706

Cisco Press

Another interesting combined use case is network optimization. While your data collection pipelines are collecting data from your network and storing it somewhere (for example, in a database), you can have a pipeline monitoring the stored data, looking for patterns or optimizations. For example, if you are collecting and storing information on the bandwidth utilization of your interfaces, it is possible that your monitoring pipeline will identify underutilized interfaces. In this case, it can trigger a configuration pipeline that alters some traffic-routing metrics to reroute traffic and better utilize your available infrastructure.

There are increasingly more uses for networking data. A practice that is becoming more common is to apply machine learning algorithms to identify patterns in networking data. Just like in the previous scenario, assume that you are collecting and storing your switches' data—this time percentage CPU utilization. You can build a machine learning model, which some consider to fall within the automation umbrella, and integrate it in a NetDevOps pipeline.

In simple terms, a machine learning model has two phases in its lifecycle: the first phase is where it needs to be trained, and the second phase is where you can use it to make predictions (called "inference"). In the first phase, you "feed" data to the model so it can learn the patterns of your data. In the second phase, you give it a new data point, and the model returns a prediction based on what it learned from past observations.

Continuing our previous example, you can train a model based on percentage CPU utilization per device family. CPU utilization is highly irregular—what is an acceptable value for a specific device doing a specific network function might not be an acceptable value for a different device in the same network. Because of this, it is very complicated to set manual thresholds. Machine learning can help you set adaptable thresholds depending on the specific device based on its historical CPU utilization.

Now what does all of this has to do with NetDevOps? You can have a pipeline that retrains a model when predictions become stale. Likewise, you can call the inference point of the model in one of your alerting pipelines and replace the static alarm thresholds.

Machine learning is starting to have many applications in the networking world—from hardware failure prediction to dynamic thresholds and predictive routing. It is important to understand that if you are using NetDevOps practices, adding machine learning into the mix is simple.

Do you have a use case not covered in this chapter? NetDevOps is only limited by what you can do with automation. So, if you can automate it, you can make it run on a CI/CD pipeline using source control and testing techniques to reap all the benefits you have learned in this chapter.

Decisions and Investments

Let's say that you love the use cases of NetDevOps because they resonate with your current challenges. So now you ask yourself, "How do I start, and do I buy something?"

In order to adopt NetDevOps, or any other technology, you will have to make several decisions and possibly some investments. This section covers the four main verticals you should consider:

- Starting point
- New skillsets
- New tools
- Organizational changes

These might seem like a lot of investments; however, considering the benefits, they are worth it. NetDevOps has some initial investments that decline over time, while its benefits grow over time, as shown in Figure 2-13.

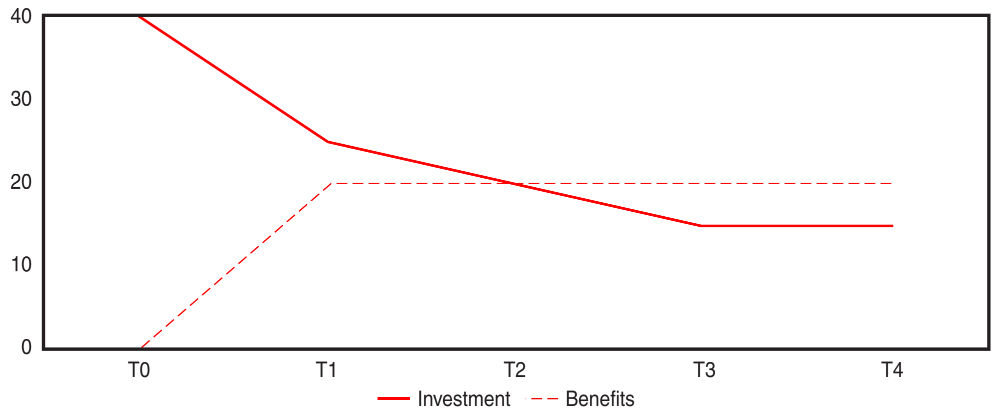

Figure 2-13 *NetDevOps Investments Versus Benefits Chart*

Because this is a fairly new field compared to others in networking, it is hard to find trustworthy resources about it. The four main verticals described in this chapter are derived from the authors' own experience in the field for the last five years working with NetDevOps. They are not industry standards.

Where to Start

When you start something new, you must begin somewhere. For example, when you are learning a new technology, you can start by reading a book or watching a video online. For some things, where you start does not matter because you'll ultimately reach the same destination; however, when it comes to adoption of NetDevOps practices, choosing the place to start is very important.

So where should you start? There is no silver bullet or a single place where all organizations should start; rather, each organization should undergo an analysis to decide what is best for its situation. This preliminary analysis should evaluate roughly where the organization is in terms of the following:

- Challenges/pain points
- Skills
- Technology stack

Why these three? There are other verticals you can consider, but evaluating these three typically results in a good starting point. Besides, the state of these three verticals is often well known to organizations, making this initial analysis cheap and fast. You do not need to produce a formal analysis with documentation, although you can do that if you wish. The result of this analysis should be an understanding of where you are in regard to these three topics.

After you have the understanding of where you are, either documented or not, you should add more weight to the first vertical, challenges/pain points. You should start your journey with use cases in mind. Do not try and embark on the NetDevOps journey because of trends or buzzwords. Solving the challenges you have identified that are affecting your organization is the priority.

Prioritize the identified challenges based on their importance for your business but at the same time measure the complexity of each challenge. The result should be an ordered list. This balance between complexity and benefit is sometimes hard to understand, so use your best judgment because this is not an exact science.

So far, you have not factored in the skills and the technology stack verticals from the analysis. This is where they come in. From the ordered list of challenges, add which technologies are involved from the technology stack and what skills would be required to solve them. Some of those skills you might already have, while others you might not. The same goes for the technologies.

Skills come in second in our three verticals. Although the next section focuses solely on skills and how they influence your NetDevOps journey, they also play a role in defining a starting point. Prioritize use cases that you already have the skillsets to implement. Technology comes next, because it is easier to pick up a new technology than a new skillset. However, this does not mean that adopting a new technology is easy, because it is not, and that is why we include it as our third factor.

For the technology stack, there will be many different nuances, and some use cases will not require all NetDevOps components to solve. For example, if your challenge is that people can make modifications to your network device configurations that go unnoticed, creating snowflake networks, and you need a way of maintaining a single source of truth, the only component you need is a version control system repository. Similarly, if your challenge is lack of speed and error-prone copy/paste configuration activities, you might just need to apply automation instead of also using CI/CD pipelines.

Understanding the minimum number of NetDevOps components you will need to adopt makes the journey to success shorter, which leads us to the highest contributing factor to the success rate of NetDevOps adoption: the ability to show successes at the early stages of adoption. Do not underestimate this. It not only motivates the teams involved, but it is a great way to show stakeholders their investments are paying off. Experiencing failures early or going for a long time without anything to show for it is the downfall of many adoption journeys.

However, you cannot really show success if you do not have success criteria. After you decide which use cases you will solve using which NetDevOps components, make sure you define what success looks like. Following up the previous example of snowflake networks and configurations changes that go unnoticed, the success criteria could be to have 80% of devices in the same functions with the same configuration, and to measure this you would audit the network. For the second example, in the slow and error-prone configuration changes environment, you could measure your current estimated time to implement a new configuration and the number of minutes of downtime caused by changes. Then you could define a success criterion of lowering this number by 20%.

Having specific success criteria allows you to show progress and improvement; however, it can also show that you are not actually solving your initial use case. This can be equally beneficial because it enables you to adjust your initial plan. In other words, failing quickly is less expensive.

To summarize, start with understanding where you are right now in terms of challenges, skills, and technology. Make identifying challenges the number-one priority because you want to ensure you are solving something relevant for your organization. Next, prioritize your challenges based on the skills you already have while minimizing the amount of NetDevOps components involved, prioritizing the ones you already have. Before you implement your NetDevOps strategy, make sure you have clearly defined success criteria and plan to show milestone successes early.

Skills

Skills are an influential factor in NetDevOps. In Chapter 1, you learned that many components of NetDevOps are not traditional networking components, and although some folks take them for granted, automation, programming, and orchestration are not an evolution of networking; rather, they are a horizontal domain of knowledge.

Most organizations have network engineers equipped with the traditional skillset of "hardcore" networking. This includes routing protocols, switching configurations, networking security such as access control lists (ACLs) or Control Plane Policing (CoPP), and all the rest. But in the same way that software developers do not know what the Border Gateway Protocol (BGP) is, traditional network engineers do not know what Jenkins or Groovy is.

The profile of a network engineer is evolving, and nowadays we're starting to see more and more a mix of networking, infrastructure as code (IaC), and orchestration knowledge. However, that might not be the case at your organization. If it is not, there are two schools of thought: upskilling/training and hiring.

You can choose to train your engineers in the skills you have identified to be missing from your "where to start" analysis, or you can hire folks who already have those skills. One option is not better than the other; each organization must make its own decision.

If you choose to train your engineers, you must take into consideration that, as previously mentioned, some of these NetDevOps skills are not a natural evolution of networking and can require considerable effort to learn. For example, software-defined networks (SDN) can be seen as an evolution of networking, and in some way they are the next networking topology. However, writing an automation playbook in a programming language is not an evolution of writing network configurations. Although the line is becoming blurry, and some terms like "network engineer v2" and "next-generation network engineer" have started to emerge, historically speaking, networking and automation have been two different domains.

Not all skills are generic skills such as automation or networking. A skill family that often is overlooked is tool skills. An engineer proficient at automation will not be an expert in all the automation tools; for example, the engineer might have worked extensively with Ansible but never with Terraform. This is particularly important if your chosen strategy is to hire, because most of the time you do not want to upskill a new hire on a new tool if you can hire someone with tool knowledge. In terms of training, this also factors in. Training someone in a tool is easier if that person already has knowledge of the tool domain. In other words, training someone in Golang is easier if they already know how to program in Python.

Another consideration is how you want to distribute your skills in each role; the upcoming section "Organizational Changes" covers how you can distribute your skills: all in one NetDevOps team or separate automation and networking teams. The number and distribution of engineers' skillsets differ based on each organization's needs.

Lastly, because you are reading this book, you probably want to become a NetDevOps engineer or transform your organization into one that uses NetDevOps engineering practices; however, not every network engineer will become a NetDevOps engineer. Expert-level networking skills are still required, and many folks may not have to take part in orchestration and automation tasks. This is a common misconception.

Tooling

Tools are an important part of NetDevOps. As you have learned, tools are enablers and not actual DevOps practices; however, some folks still commonly label tools as DevOps. Nonetheless, tools will still represent a big part of your investment. Not only because of their price but also because of tool-specific skills and knowledge. After you and your organization acquire these skills, changes result in added effort and cost.

Within the NetDevOps umbrella, you can separate the tools into the following different categories:

- Infrastructure as code
- Continuous integration/continuous delivery (or deployment)
- Source/version control
- Testing
- Monitoring

The following list provides examples of tools in each category. Note that this is not an exhaustive list; each category has a plethora of tools to offer.

- IaC
  - Ansible
  - Terraform
  - Pulumi
- CI/CD
  - Jenkins
  - GitHub Actions
  - AWS CodePipeline
- Testing
  - EVE-NG
  - GNS3
  - Cisco Modeling Labs CML
- Source Control
  - GitHub
  - GitLab
  - Bitbucket
- Monitoring
  - Datadog
  - ELK stack
  - Splunk