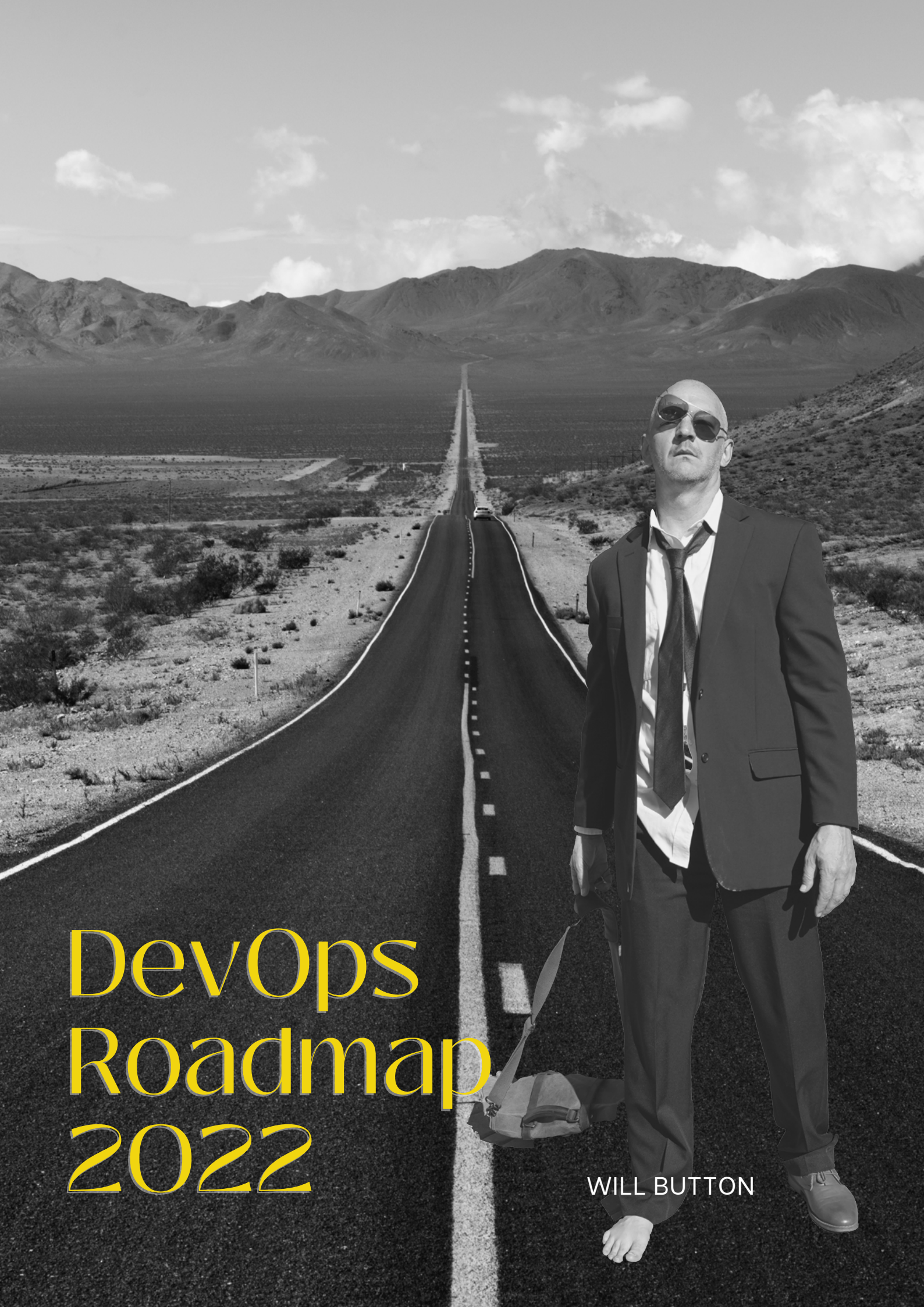

DevOps
Roadmap
2022
WILL BUTTON

01

Start Here

Welcome, adventurer.
I'm going to assume that if you've found your way to my landing page, filled out your email address, clicked the link, and downloaded this guide: you know what DevOps is.

So we're skipping that, m-kay?

My assumption is you want to know how to learn it. It's a huge topic. I've treaded these waters for decades and now I want to help you succeed in them.

Spoiler alert: you will never learn it all.
The more you learn, the less you realize you know.

Spoiler alert #2: You have to be comfortable learning every day for the rest of your career. If that doesn't sound appealing | fun | enjoyable | tolerable. Stop right here. I'm going to say you a ton of work and lost time. This is not the career for you.

For everyone still with me, this guide is a

Choose Your Own Adventure

That means you can look at the sections and choose whichever one interests you.

In each section, you'll find some broad goals and objectives to help you guide *what* you should be learning and *how* to gauge when it's time to move to the next section. At the end, you'll find some guidelines on which section to tackle next, based on how that section relates to the one you just finished. So no matter where you start, you'll eventually cover all the sections.

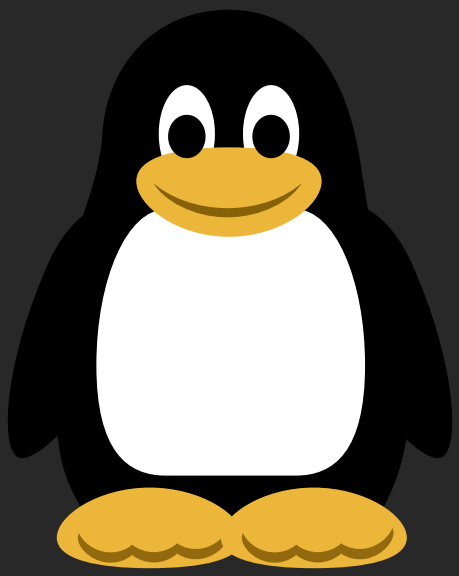
As your skills grow, put them to the test. Using virtual machines, cloud services, or old hardware you have access to: make sure you're getting firsthand experience with the tools you are learning about.

Let's do this.

Table of Contents

SECTION 02

Linux



The main objective for Linux is to be comfortable with the administration, configuration, and operation of the OS. You will spend a lot of time working with Linux so any effort spent here will pay dividends later. As an added bonus, learning how Linux operates takes a lot of mystery out of "how things work".

You don't need to become an expert but you should be comfortable with the tasks here. If you need to Google or use the man pages for syntax help is perfectly acceptable. The key thing is to know what tool to use.

You can use any Linux distribution you like, though I would recommend CentOS or Debian as these are the ones you'll encounter most frequently. If you have never used Linux before, Ubuntu is very user friendly.

Skill Objectives:

- Install/configure OS
- Using BASH
- Starting/stopping/creating services
- User management
- The Linux filesystem
- Text manipulation (awk, sed, grep, wc, echo, cat, tail, etc...)
- Process monitoring with top, ps, lsof
- Networking and network configuration
- System performance
- Using screen or tmux
- Logging in locally and remotely
- Firewall management
- Install/configure applications using the OS package manager (apt or yum)
- POSIX basics
- File permissions

Where to next?

Choose from the following related topics for the next stop on your journey

Infrastructure as Code

Now that you understand Linux a little better, you may be ready to learn about IAC. Building and maintaining servers can get complex. IAC allows you to define those servers as code, then let automation take care of the dirty work for you.

Scripting

There are a lot of tasks to do in Linux administration. Combine that with multiple servers and you've created a fair amount of work for yourself. Scripting can make that a little easier by automating many of those tasks.

AWS

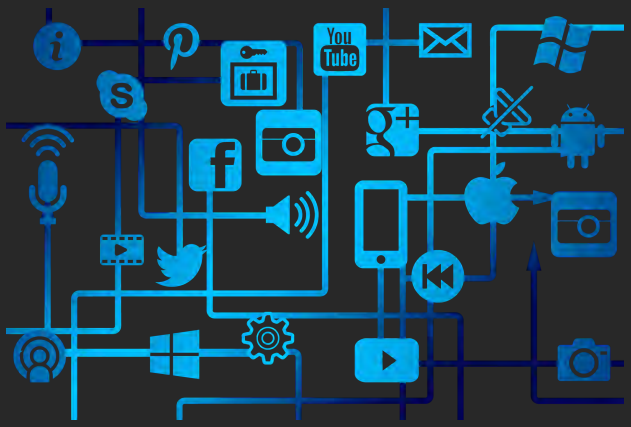
With a better understanding of the OS environment, learning how to build and manage servers in the cloud may be a great next step.

Networking

All of those servers need to talk to each other right? Networking is the key to making that happen. Learning networking gives you the knowledge to build super-highways instead of dirt roads for your digital citizens.

SECTION 03

Networking



No, not the social event where everyone stands around staring at their drink, except for those few people who actually seem to be enjoying themselves. Networking refers to the protocols and models that allow computers to talk to each other. If you've ever connected to a WiFi router, you've experienced networking.

The main objective for this section is to understand the layers and models of the network stack, and how they interoperate with each other to provide secure communication where needed and prevent access where it's not needed.

Key concepts to understand for networking:

- OSI 7 Layer Model
- Network Topology
- IP Addressing and subnets
- TCP and UDP ports
- Well known ports (<1024)
- DNS
- DHCP
- Routing
- Firewalls
- Network Address Translation (NAT)
- Common protocols: TCP/IP, ARP, HTTP(S), FTP, SMTP, SSH, POP3
- Network troubleshooting tools: ping, nmap, traceroute, netstat, telnet, iptables

And while not specifically networking, I think this is the right place to focus on the public-facing aspects of your architecture using DNS:

- CNAMES
- DKIM, SPF, and DMARC records
- Website caching using CDN
- Domain Keys

Where to next?

Choose from the following related topics for the next stop on your journey

Security

All of this talk of networking makes it possible for bad actors to get answers to the wrong questions, right? Let's go take a look at security to learn how to close those security holes.

Infrastructure As Code

Subnets, and masks, and firewalls... Oh my! That is a lot to take care of so let's go learn about Infrastructure as Code to see how much of this we can abstract and automate.

Kubernetes

So you want to build a network inside a network? Me too! K8S has some interesting ways to scale your applications without using your physical network. It could be an interesting next-stop on your journey.

Observability

Someone should probably keep an eye on this networking stuff, right? Seems pretty important so maybe implementing some monitoring, logging, and alerting is a good place to focus next.

SECTION 04

Security

Security doesn't really need an introduction. You know you have to be secure, but what does that really *mean*? What are the *things you do* to be secure?

That's what you'll focus on in this section. Security is everyone's problem and while you won't know how to address every security gap, you'll know how to see one when it exists.

I break security down into 3 areas: user security, application security, and network security. The following objectives should help you wrap your head around them.

User Security:

- User management
- Password policies
- File permissions
- Sudo access

Application Security:

- Dependency updates
- Secrets management
- Vulnerability scanning
- Docker container security

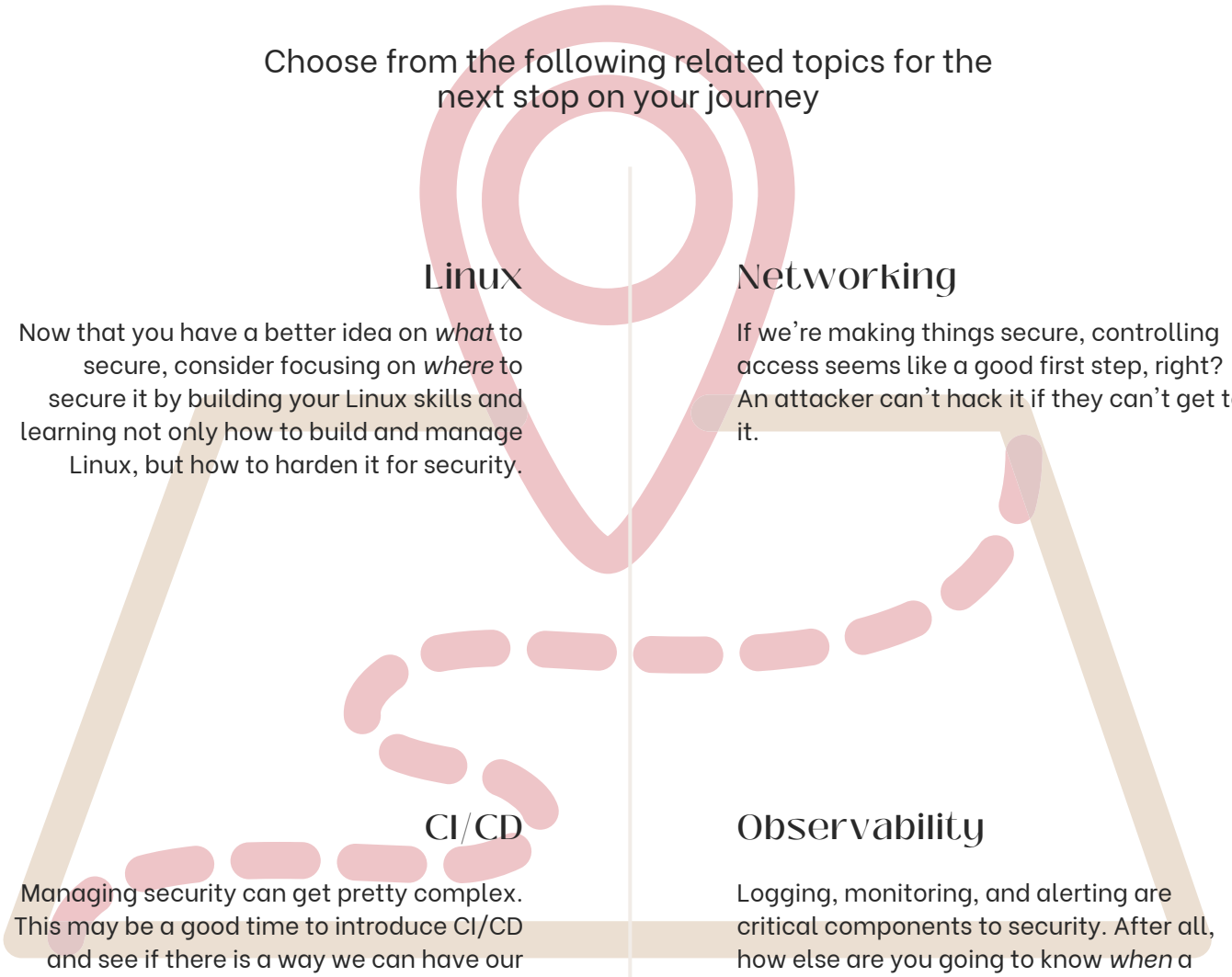
Network Security:

- Open ports
- Firewalls
- Logging
- Pen-testing

Despite the fact that there are only a few bullet points there, this topic is huge. One good way to approach this is after learning about the objectives above, come back and re-address this topic after each of the other sections.
Bonus objective: Disaster Recovery.
You *will* be hacked. How will you recover and keep your company operating when that happens?

Where to next?

Choose from the following related topics for the next stop on your journey



Linux

Now that you have a better idea on *what* to secure, consider focusing on *where* to secure it by building your Linux skills and learning not only how to build and manage Linux, but how to harden it for security.

Networking

If we're making things secure, controlling access seems like a good first step, right? An attacker can't hack it if they can't get to it.

CI/CD

Managing security can get pretty complex. This may be a good time to introduce CI/CD and see if there is a way we can have our security policies applied and verified with every deployment.

Observability

Logging, monitoring, and alerting are critical components to security. After all, how else are you going to know *when* a security incident happens?

SECTION 05

Infrastructure as Code

Giggity.
That's the best way to summarize Infrastructure as Code (IaC). If you've been following along in the other sections, you've realized there are *a lot* of moving pieces here.
So how do you ensure that the firewall settings you applied to that Linux server 2 years ago are still there?
How do you verify that any new Linux servers get the same firewall policy?

You don't. You let code do the work for you, happily making changes to anything that doesn't meet your specifications. If only raising kids were that easy...

The objective for this section is simple but not easy, Think of every application, network, server, or other device you built, managed, and configured in this guide. IaC allows you to reproduce it automatically, with no chance for human error. It also allows you to do it an infinite number of times with the exact same results.
Pick a tool, really any tool: Ansible, Puppet, SaltStack, CloudFormation, or Terraform. I like Ansible personally, YMMV.

Learn about the tool, then apply that knowledge to the tasks you are doing in this guide. Every file you've edited, server you've built, firewall you've configured: do it all again with nothing but your IaC tool.

While you are at it, make sure your IaC lives in a Github repo. This sets the stage for you to use CI/CD to manage your infrastructure. Remember though: no secrets or passwords can be stored in your git repo.

Where to next?

Choose from the following related topics for the next stop on your journey

Linux

A significant portion of your infrastructure will be Linux servers. This is a good time to brush up on those Linux skills with a focus on identifying what parts we can implement via IAC.

Networking

In today's world, a lot of our networking is virtualized in the cloud. That means we can define our networks as code and allow our IAC to manage them for us.

Security

An important component to IaC is "*who needs access to what*", so maybe a quick study of security can help answer that question and provide some guidelines on how to implement it.

CI/CD

It's Infrastructure as *Code*, right? If it's code, that means we can automate the deployment. And if it's deployed, we can use CI/CD. That makes CI/CD a great topic to study next.

SECTION 06

CI/CD

FOR THOSE ABOUT TO DEPLOY

CI/CD or Continuous Integration / Continuous Deployment is most commonly what people think of when they think of DevOps. That perception isn't unjustified: it's usually the "thing" that produces a visible artifact of the DevOps process.
As you are learning in this guide though, it's only one piece of many under the DevOps umbrella.
Think of a vending machine: you may think of the spiral arm that drops your candy bar as the CI/CD component but there are many other pieces that will prevent delivering the expected experience in addition to the arm. Hopefully, this guide is helping you understand that.

The objective of this system is to build something. A tangible artifact, using CI/CD such that the artifact is built, configured, and delivered automatically.
The best way to learn the concepts involved is to simply do it. You'll need to pick a tool: Jenkins, Gitlab, Circle CI, Github Actions, or any one of the hundreds more to choose from. I recommend Circle CI or Github Actions unless you have a specific reason for choosing otherwise.
For the sake of efficiency, you may want to build and deploy a server as described in Section 05, and use CI/CD in this section to automate that process. Or you can deploy one of the many To-Do style apps available from Github.
Objectives:

- Create a Github repo
- Run linting when code is committed
- Run tests when a pull request is opened
- Build the artifact when a pull request is merged (i.e. Docker image, compile, bundle)
- Deploy code changes from main or master branch when a release tag is created
- Send notifications if any pipeline stage fails

Where to next?

Choose from the following related topics for the next stop on your journey

AWS

Now that you have a better understanding of deploying software, it's a good time to learn about where that software will be deployed. AWS is likely going to be part of that equation.

Kubernetes

Just like AWS, at some point Kubernetes may be part of your environment. Learning the fundamentals of K8S will give you a better understanding of where you deploy your applications.

Infrastructure as Code

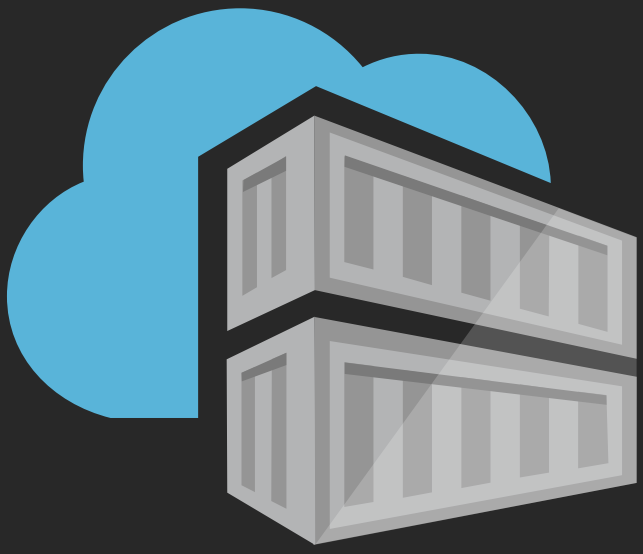
One of the really cool implementations of CI/CD is you can use it to build, manage, and maintain your infrastructure using the same CI/CD concepts you use to deploy applications.

Observability

An important part of deploying software is tracking the status of deployments and performance. Observability provides that framework and makes it an excellent choice for you next study topic.

SECTION 07

Containers



If you haven't figured it out yet, containers are an integral part of modern software engineering. Fortunately, you don't have to reach expert-level knowledge to apply this in your day to day DevOps activities.

The reason containerization is so popular is it allows us to consolidate and isolate the resources needed by our application, scale faster and easier, and take advantage of compute resources across our environment.

To state it another way, containers allow us to run our apps, focusing only on the app and its dependencies without having to manage an Operating System, hardware, networking, configuration, and updates across every single app we support. We can do that at the host OS level and allow our containers to leverage that work as needed.

Your primary goal for learning containers should revolve around the following:

- Building efficient, secure images
- Tagging images to clearly convey what each image does (for example, what is the difference between myimage:latest and myimage:v0.0.1
- Implement best practices in your Docker images (see the Docker docs)
- Mounting volumes to persist data across restarts
- Exposing ports & communication between containers
- One container: one process

A great road-test for your Docker knowledge is to use one of the many repos on Github as the source application for your container. I.e.- build a working image using the code in the repo. Then deploy it using what you learn in the CI/CD section.

Where to next?

Choose from the following related topics for the next stop on your journey

Linux

Your containers rely on the container host for operation, which is likely Linux. That makes Linux an excellent choice to study next if you haven't already covered that topic.

Kubernetes

Learning K8S is a great way to learn how containers operate in concert at scale. K8S is fully implemented using containers, not only for the applications you deploy on it, but for core K8S functions as well. A worthy next topic.

Programming Language

Many of your container use cases involve executing the code written by your development team. Time spent learning how to write, build, and run applications can further boost your container knowledge.

Scripting

It's fairly common to implement some scripts in the creation and maintaining of containers. That makes scripting an excellent choice for your next area of study.

SECTION 08

AWS

While there are many cloud providers to choose from, I recommend starting with AWS. It's the industry beast and the most widely used of all the cloud providers. If you have a strong preference for a different provider, by all means: go learn it.

In any case, many of the skills you learn for AWS directly transfer to Azure and GCP so it's not like you will have to start from scratch when you learn AWS but then land a job working in Azure.

Learn the fundamentals, and build from there.

AWS (and all cloud providers) have hundreds of features and services. You can drive yourself crazy trying to learn them all. Fortunately, we're going to focus on the fundamentals. By doing so, you'll be productive on the platform and armed with the skills to evaluate the pros and cons of the other services on an as-needed basis.

Objectives:

- VPC - absolutely mission critical
- IAM - access to everything is governed by IAM
- Security Groups - used to control access at a network level
- EC2 - running virtual machines in AWS
- Load balancers - the key to successful highly available, fault tolerant applications
- RDS - managed database services
- Lambda - Serverless applications in AWS
- Fargate - A great way to start using containerized applications
- S3 - storage, not to be confused with:
- EBS - volumes for your EC2 instances
- KMS - a place to keep secrets like API keys, passwords, etc...
- Cloudwatch - monitoring for all things AWS
- Billing - this stuff can get expensive

Where to next?

Choose from the following related topics for the next stop on your journey

Linux

Your AWS infrastructure will likely consist of many Linux servers, so why not get a head start on that and make Linux your next area of study?

Scripting

With so many tasks, dependencies, and interactions in AWS, learning scripting will be time well spent to automate many of the routine tasks needed.

Infrastructure as Code

Oh man, there are a lot of components in AWS, right? IaC is a great way to keep track of them all in a predictable, consistent way.

Observability

With so many moving pieces, you are going to need a way to view things from a 10,000 foot view and drill in as needed. Observability provides the framework to allow you to do just that.

SECTION 09

Kubernetes

Who doesn't love Kubernetes? Evidently no one according to #techtwitter!
Seriously though, if you jumped to this section first: you may want to rethink that strategy. K8s is a beast of a topic and makes more sense after you have a grasp on the other topics in this guide.

One key objective for learning Kubernetes is learning *when you shouldn't use Kubernetes*. It's cool and flashy and popular, but can become an unwieldy beast that may not be needed at all for a lot of environments.
With great power comes great responsibility

Oh... still reading, eh? Ok- here's what you should focus on:

- Pods - what are they and how to use them
- Services - what role do they play in k8s
- PV & PVC - making data storage available to your pods
- Deployments - a specific k8s construct (differing from the "d" in CI/CD)
- Managing secrets in k8s
- Building your own k8s cluster
- Monitoring with Prometheus and Grafana
- Using ingress controllers
- Managing apps using helm

You don't need to go deep on these initially, but you do need to know what they are:

- CoreDNS, k8s networking, k8s API, control plane, nodes, kubelet, etcd

Where to next?

Choose from the following related topics for the next stop on your journey

Networking

Properly implemented networking is critical to managing a secure K8S implementation. That makes networking an excellent topic to study next.

Containers

One hundred percent of K8S is implemented via containers. If you're not comfortable building and managing containers, it's an excellent topic to dive into next.

Infrastructure as Code

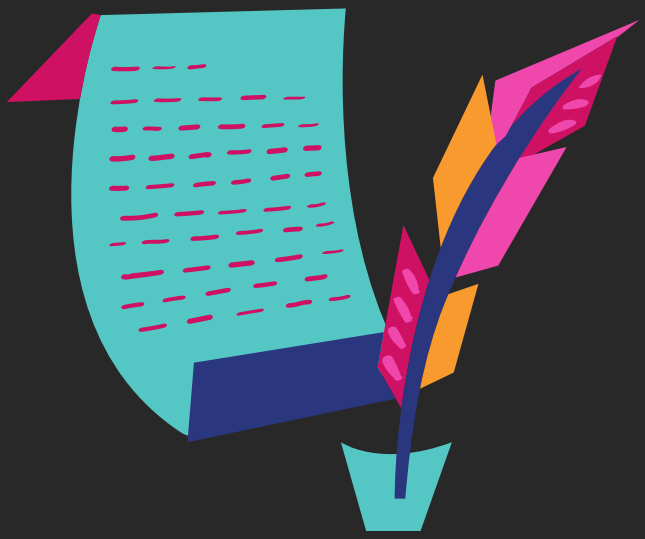
There are a lot of pieces to K8S, right? IaC provides a framework for managing those pieces. Study this next to learn how to build and deploy K8S as code, not mouse clicks.

Observability

With so many moving pieces in K8S, observability is critical for understanding what's going on inside and where your bottlenecks are. Plus, K8S has some common integrations just for this task.

SECTION 10

Scripting



Scripting is a must-have skill for DevOps and sysadmins alike. Scripting is your go-to resource when you need to make repetitive changes across multiple systems. Think about tasks like:

- applying a security patch to 1000 servers to close a security hole
- copying millions of files from one storage location to another
- digging through log files to identify a specific scenario

It also sets the stage for you to learn programming, which you will eventually do. Even after you know a coding language, you'll still use scripting for those quick and dirty tasks.

You can choose from a few different options for scripting: BASH, Python, Javascript, or Ruby.
I personally recommend BASH: it's guaranteed to be on every system you ever touch. Python is a great choice also, and can double as your programming language when you reach that point. I advise against Javascript. Its asynchronous nature is not suited for our use case. Ruby is fine but not common. I'd only recommend it if you knew you were going to be working in a Ruby shop.

Objectives:

- Making scripts executable
- Capturing input from keyboards and computers
- Reading/writing to filesystems, networks, and (in some cases) databases
- Providing script feedback (output status) and reacting accordingly
- Executing scripts on remote computers
- Using conditionals, variables, logic, functions, and proper syntax

Where to next?

Choose from the following related topics for the next stop on your journey

CI/CD

Almost all CI/CD pipelines have some scripts in them. That makes CI/CD an excellent choice of study next, to learn where all these scripting skills come into play.

Infrastructure as Code

Managing IaC requires a lot of moving pieces. Put your scripting skills to the test by tackling this section next and using your scripts as the glue for your IaC.

Containers

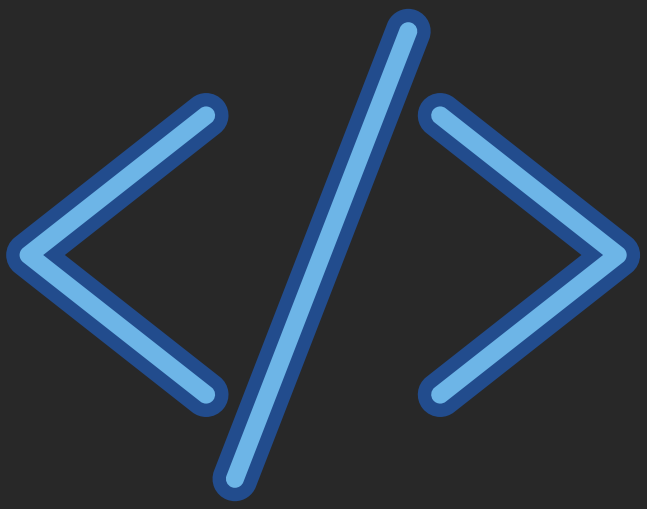
It's common to see some scripts used in the creation of containers, so it's an excellent place to study next and learn how to use your scripting skills to build fast, reliable, secure containers.

Programming Language

If you enjoyed the power scripting gives you, you may want to dig further into this topic by learning a programming language to see exactly how deep this rabbit hole goes.

SECTION 11

Programming



Since DevOps is a collaborative role between Development and Operations, it makes sense that you will need at least a fundamental understanding of programming.
In your career it will not be uncommon to debug code, contribute to code to help your Development teams implement DevOps practices, review code, and even create your own applications.
Many aspiring DevOps engineers get nervous when it comes to coding. In this section, you'll learn the basics, giving you the confidence to take on more challenges and responsibilities.

When choosing a language, I think there are three top candidates:
Javascript (Node.js): It's 100% guaranteed you will work with JS developers. It's in use in every company.
Python: The tried and true old-timer in sysadmin circles. It's also commonly used in server applications as well.
Go: Unlike the other two, Go is a compiled language and is gaining much adoption in the DevOps space. A lot of the tools you will work with on a daily basis are written in Go.

Objectives:

- Language syntax
- Workflow (i.e. git workflow)
- Variables, types, flow control, and logic
- Reading and writing data to file objects and storage objects like databases, message queues
- Packaging and deploying your code
- Debugging
- Handling errors
- Writing tests

Where to next?

Choose from the following related topics for the next stop on your journey

Infrastructure as Code

If you're learning to code, you may as well apply it to your own job, right? IaC does just that, allowing you to define and build your infrastructure using the coding skills learned in this section.

CI/CD

The code you've learned to write has to be deployed somewhere to do anything, right? That makes CI/CD an excellent choice to study next to learn how to build and deploy applications at scale.

AWS

One of the interesting uses for programming is in your AWS environment. AWS has a complete SDK for many languages, allowing you to leverage those programming skills for your own good (or evil, I'm not judging).

Observability

You've built that killer app, but what's it doing? That's a critical question. One that studying observability will help you answer.

SECTION 12

Observability

If a tree falls in the forest when no one is around, does it make a sound? If a server crashes when no one is around, does it make a sound?
I don't know about the former but I can say conclusively that latter will never happen. Not due to some strange laws of nature, but because in this section you will learn how to implement logging, monitoring, and alerting to ensure no server crashes without an alert being raised and the proper on-call resources notified.
Observability is in fact, the eyes and ears of your team for your applications.

Again, there is a bit of flexibility in the exact tools you learn here, but the concepts are the same. There are three areas to focus on:

Logging: applications create logs, which are useful in understanding how the system is performing. It's important to get the logs into a centralized server so your team can view them without needing to log into each server. Logstash and Cloudwatch Logs are great starting points for this.

Monitoring: We need a way to visualize and track performance metrics to identify capacity, load, and trends. Check out tools like Prometheus, Grafana, and Datadog. Don't forget application-specific monitoring tools like Jaeger, Scout APM, and New Relic.

Alerting: Using the data from the other two categories, we can create alerts to let our team know when things have gone horribly wrong. That's a great thing. Your monitoring tools probably have this capability so be sure to use it. Also familiarize yourself with PagerDuty.

Where to next?

Choose from the following related topics for the next stop on your journey

CI/CD

CI/CD is one of those things that benefit from great observability. Knowing when deploys happen and when they failed is good to know, so learning the ins and outs of CI/CD can be a great topic for your next study session.

Infrastructure as Code

With all of this monitoring in place, learning how to respond to it in an automated fashion can dramatically improve your reliability and uptime. That involves IaC, which can make it a great topic to study next.

AWS

There are many places in AWS to put your Observability skills to the test. Head on over to this section to learn more about AWS and how to leverage the skills learned here.

Kubernetes

Much like AWS, your K8S environment can put your Observability skills to the test. Learning how to build, maintain, and monitor K8S can be a great follow up to this section.

Reach out!

PO Box 18733 Fountain Hills AZ 85268
devopsfordevelopers.io
hello@devopsfordevelopers.io
@devopsfordevelopers