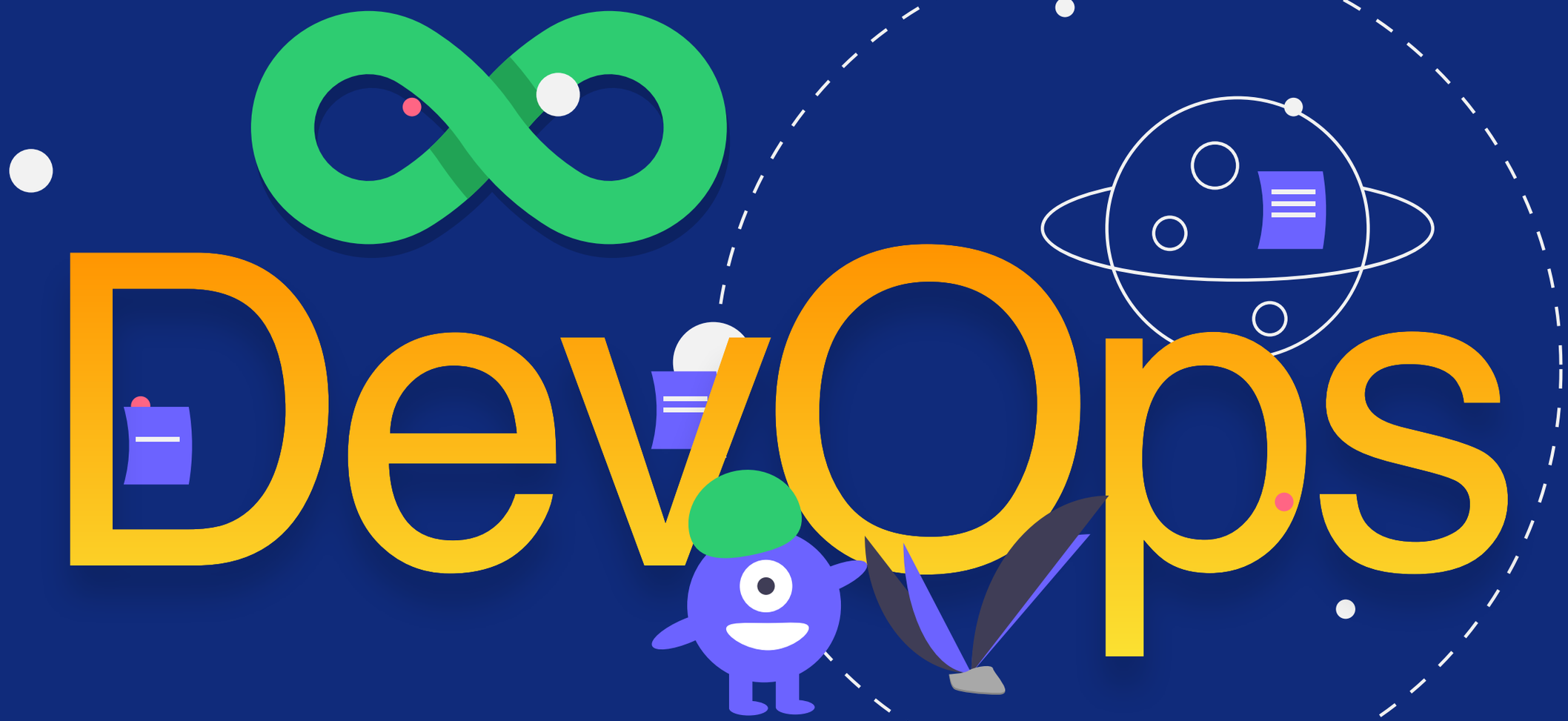
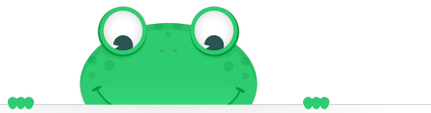
leapfrog



DevOps

HANDBOOK



Credits

Authors Pratish Shrestha

Sparsha Dotel

Designers Swapnil Acharya

Abhash Bikram Thapa

Reviewers Suja Manandhar

Anish Krishna Manandhar

Special Thanks

Chris Sprague (CEO Leapfrog)

All Leapfroggers

Abstract

You might have heard of the term “DevOps” but never knew what it meant. Does it signify a person, a role, or a process?

This handbook will lay down the foundation about what DevOps is and how you should be following a DevOps culture. It will help you understand various aspects under a DevOps model that will guide you on your journey.

Who is this handbook for?

This handbook is primarily written for developers and engineers who have a certain level of experience in the field of software development. However, it is also intended for anyone interested in knowing about the DevOps culture.

How should you use this handbook?

Let this handbook be the entry point for you in learning about the DevOps culture. This handbook only touches the DevOps model at a surface level, but it will provide you with the overall knowledge of what you should be doing to make your workflow better. Once you get familiar with these topics, you can dive deeper on your own by continuous research and implementation.

What is DevOps?

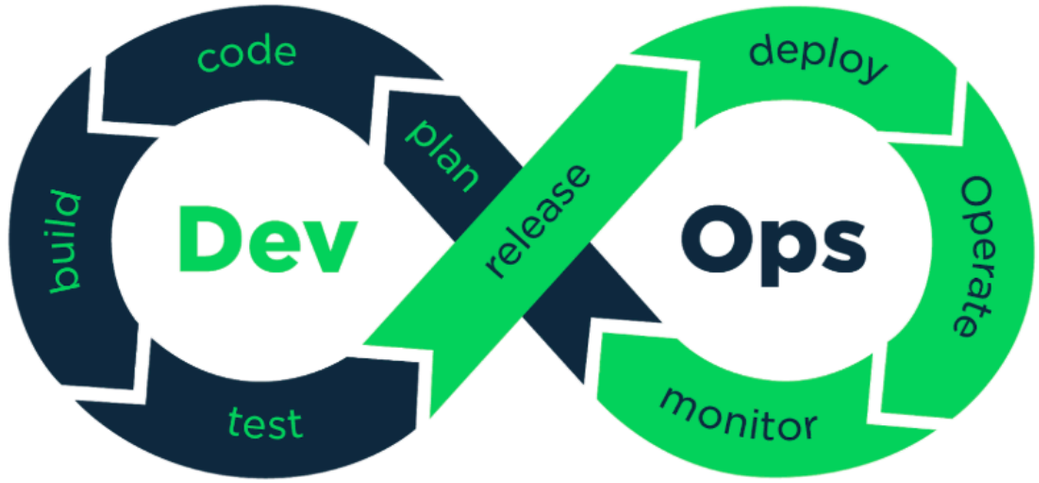
“Dev” refers to software application development, and “Ops” refers to IT operations.

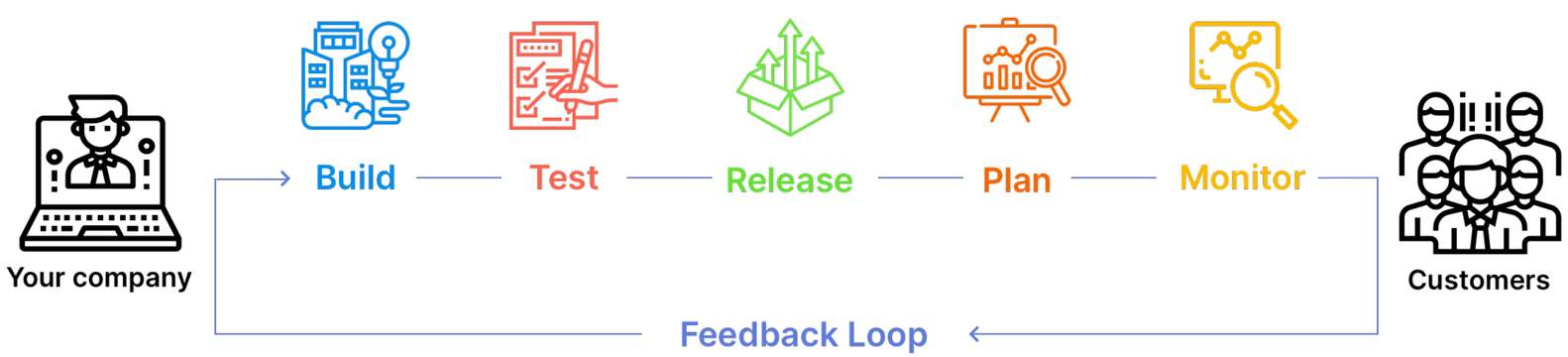
To simply put it, DevOps is a culture, not a role! The whole company needs to be doing DevOps for it to work.

AWS defines DevOps as the combination of practices that increases an organization’s ability to deliver applications and services at high velocity. This speed enables organizations to serve their customers better and compete more effectively in the market.

Under a DevOps model, the development and operations team are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

Moreover, they use a technology stack and tooling, which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks like deploying code or provisioning infrastructure that normally would have required help from other teams, and this further increases a team’s velocity.





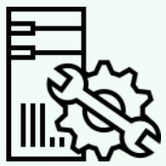
How do we adopt a DevOps model?

Teams need to follow a few key practices to streamline the software development and infrastructure management process. Most of these practices are accomplished with proper tooling.

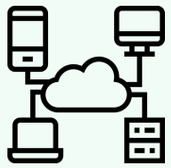
In this handbook, we have divided the process into 6 such practices.

Infrastructure as Code

Containerization



Config Management



CI/CD

Logging and Monitoring

Using the Right Service

Infrastructure as Code

The applications we create need machines to run on. You have to set up your own servers, add the required operating system, connect it to a network, and make needed adjustments. As a result, it becomes a lengthy and labor intensive process. But now, due to the availability of various cloud services like AWS, Azure, and Google Cloud Platform, all provisioning can be done with a click of a button.

While it sounds simple, we do not suggest clicking buttons to accomplish these tasks. Why? Because button clicking is:

- Error-prone
- Not versioned
- Not repeatable
- And, not testable

Even if the process above succeeds, you will still have to create your DEV environment, then QA, UAT, and finally, PROD environments. This gets very tedious and annoying very quickly.

This is why we need infrastructure-as-code.

As a best practice, infrastructure-as-code mandates that, whatever work is needed to provision computing resources, it must be done via code only.

So how do we accomplish this?

Simply, use **Terraform**.

You write your infrastructure state in Terraform, store it in your source code control, and execute it to provide all the resources needed.

In a nutshell, you write code to setup your servers.

There are other tools like CloudFormation, Chef, Puppet, etc. that can be used for the same purpose. However, currently, we recommend Terraform as it is cross-platform, meaning it works for AWS, Azure, and GCP. In addition, it is easier to learn than others.

Containerization

Just because your application code runs in your local machine does not always mean that it will work on your production servers, or as a matter of fact, not even in your team member's machine. Machines will always differ. You may have different operating systems, conflicting dependencies or different versions of those dependencies.

This is where Docker comes in. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. You need to package your code for easy deployments and be confident that it runs in any machine.

Now there might be a bit of confusion regarding the docker containers and docker images.

A docker image is a combination of binaries, packages, and possibly source codes that are needed for an environment to run a program. A container is simply an instance of the image.

If the image is a game disk, a container is the installed game that you play.

So, where do you run your containers?

It's always a good idea to use a container orchestration service instead of spinning up your own Virtual Machines. A container orchestration service helps in many tasks, such as:

- Provisioning and deploying containers
- Scaling up or removing containers to spread application load evenly
- Allocation of resources between containers
- Load balancing and service discovery between containers
- Health checks and monitoring

To run these, there are various services in the market, such as AWS Fargate, Google Cloud Run, Amazon EKS, etc. The benefits of using these are that; you don't have to worry about scaling your servers, it does it for you. It's reliable and has auto-scaling features which will save you a lot of headaches in the future.

AWS Fargate

Amazon EKS

Cloud Run

Config Management

An app's config is everything likely to vary between deploys (dev, UAT, staging, production, etc.). This includes:

- Connection strings to the database such as database username and password,
- Credentials to external services such as Amazon S3
- Per-deploy values such as the domain names for the deploy

Apps sometimes store config as constants in the code. However, as good practice, config should be strictly separated from the code.

An app should get its config from environment variables. Environment variables or Env vars are easy to change between deploys without changing any code; unlike config files, there is little chance of them being checked into the code repo accidentally.

To achieve this, you should first modify your code to initialize your configuration from the environment variables (if you haven't already). Use .env files for local development, but do not check it into the code repo. Always keep `.env` files in `.gitignore`.

For deployments, use tools such as HashiCorp Vault or AWS Secrets Manager. These services guarantee that your secrets are secure and easy to manage.

CI/CD

CI/CD refers to the combined practice for Continuous Integration and Continuous Delivery. Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which we run automated builds and tests. The key goals of continuous integration are to:

- Find and address bugs quicker,
- Improve software quality,
- Reduce the time it takes to validate and release new software updates.

Whereas, Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

Essentially, deployments should not cause so much hassle to the team. You should not be taking an entire day to release your changes. It should be done in a matter of minutes.

As soon as a change is pushed and merged, it should run all test cases and should be deployed to the appropriate environment. Everything should be automated with the help of CI/CD tools.

There are various tools in the market right now. You can choose from anyone from them, namely, Travis CI, Circle CI, Github Actions, Azure DevOps, etc.

Travis CI

Azure DevOps

GitHub Actions

circleci

Logging, Monitoring, and Alerting

Logging

Let's talk about logging. Logs provide visibility into the behavior of a running app. Logs in their raw form are typically a text format with one event per line.

Now, an app should never concern itself with the storage of the output stream. That is, it should not attempt or manage logfiles. Instead, each running process writes its log stream to stdout. During local development, the developer will view this stream in the foreground of their terminal to observe the app's behavior.

In staging and production environments, each log stream is captured and sent to a centralized logging system. This is where services like Elastic Stack and Sentry come in.

Remember containers in the earlier section? Let's say your app is running on multiple containers, scaled horizontally, and kept in front of a load balancer. Basically, you have multiple instances of your app, separately logging into its own log stream. How do you find the data at a given time from these streams? Do you look into all of the instances to find your log? No.

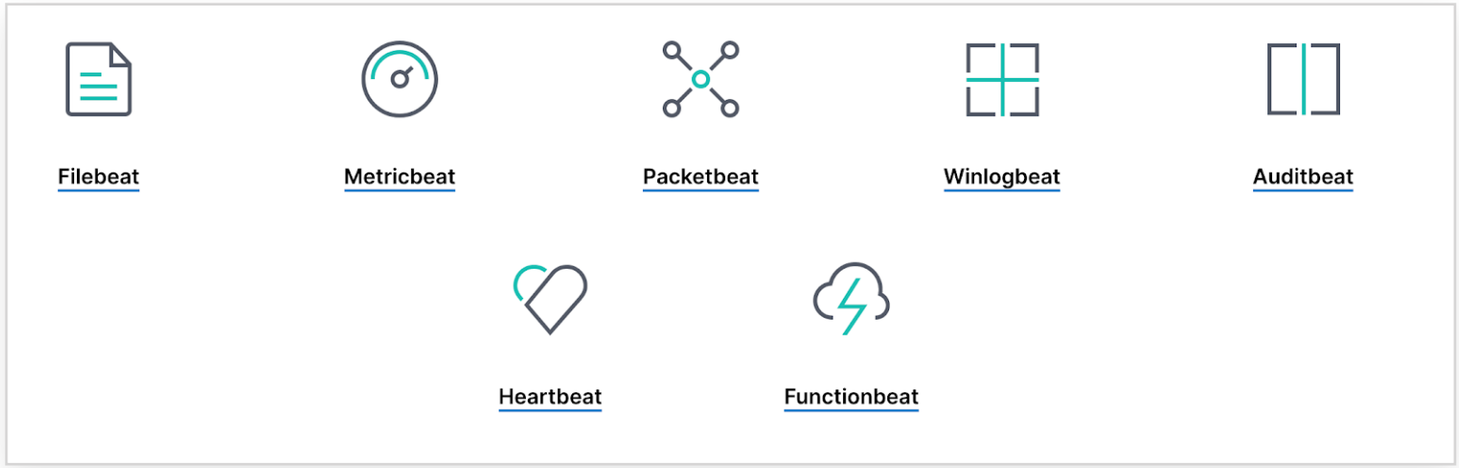
You use something like **Elasticsearch** to centralize all these logs. Essentially you ship all the logs from these containers to a single elasticsearch database (Elastic stack provides various **Beats** for this purpose). Then, you use a dashboard like **Kibana** to query your logs. Since all your logs are aggregated into a centralized location, it's easier to search and locate your logs.

Also, services like Sentry helps to track your error logs. Each time an error occurs in your application, it's logged in the Sentry dashboard. Moreover, the developers are notified as soon as an error occurs in the application.

Monitoring

Now, let's talk about monitoring. You should know everything that is happening in your system. As soon as there is a problem, you and your team should get notified about it. Active monitoring becomes increasingly important as services must be available 24/7. Creating alerts or performing real-time analysis of this data helps organizations proactively monitor their services.

There are multiple tools that you can use for that. The easiest way to get started is NodePing. With this, you can create various checks (HTTP/S checks, Port Checks, SSL certificate expiry checks, etc.) to get an understanding of any sort of malfunctioning in your system and get an email alert when there is a problem in your system.



NodePing

NodePing will only check if your applications are up or down. However, you need to view your system metrics, such as CPU usage, RAM usage, Disk IO usage, etc.

For this, you can use tools like Elastic Stack again as mentioned above.

Elastic Stack provides various plugins called "Beats", specifically "MetricBeat" for this particular use case.

Elastic Stack is just one of many tools that you can use. There are other services such as Prometheus, Zabbix, Grafana, Datadog, etc. Use what best fits your needs. Generally, you should know what is going on in your system and be prepared if something ought to go wrong.

Elastic Stack is just one of many tools that you can use. There are other services such as Prometheus, Zabbix, Grafana, Datadog, etc. Use what best fits your needs. Generally, you should know what is going on in your system and be prepared if something ought to go wrong.

Grafana

Prometheus

DATADOG

Choosing the right tools and services

Often we are asked, “What does a project truly need?”

Much depends upon the project itself.

There are hundreds of tools and services in the market right now, and it can be overwhelming. You might be learning one tool and another shiny new service might just pop out which is far better than the tool you are using. That’s how technology works.

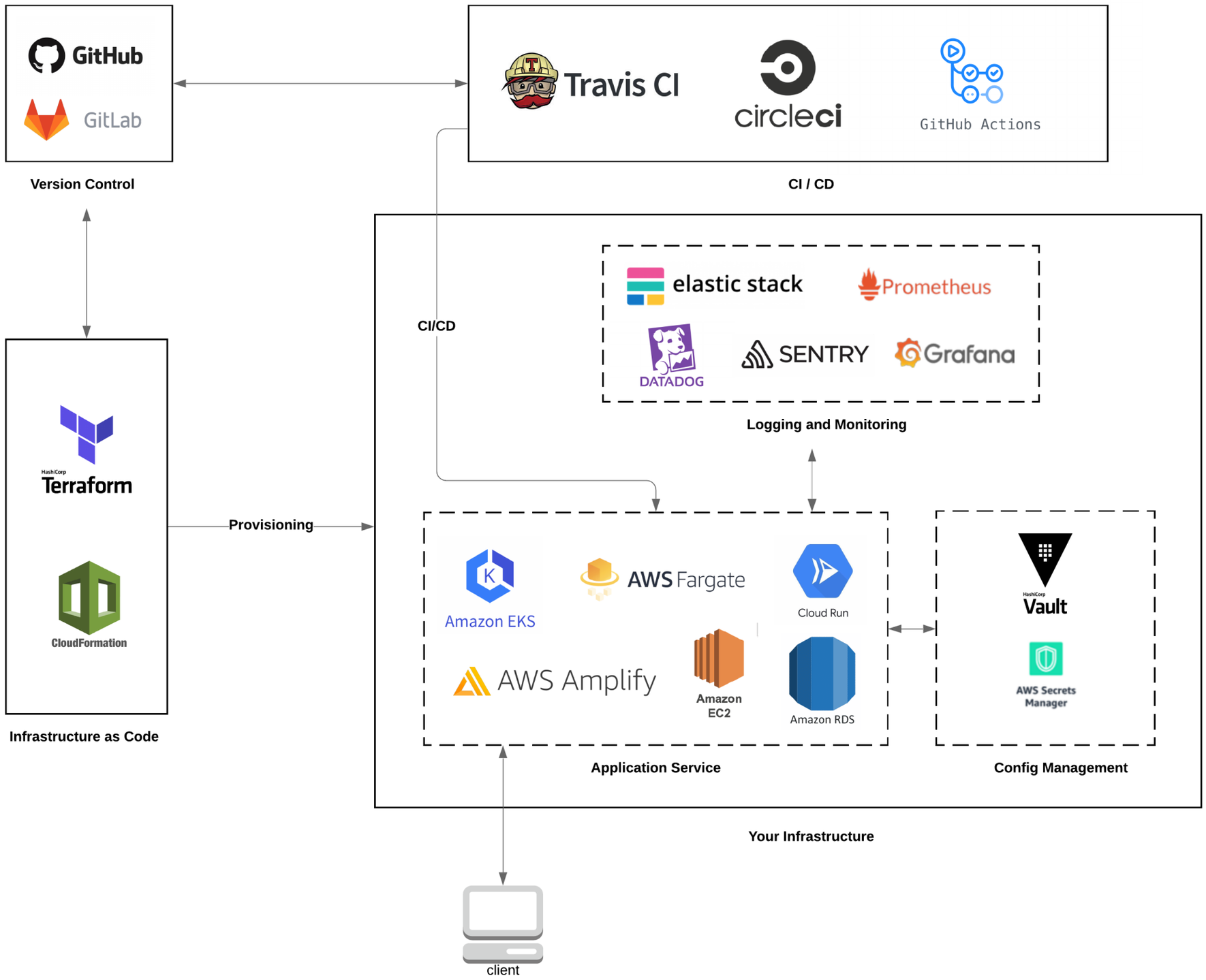
However, tools are there just to make your life easier. Know the basic principles of what you want to achieve and how it can be done. After that, you can choose any tool that is available. As you go on and try out various tools and services, you will have your own favourites and your own dislikes.

Use what makes sense to you and the project.

Don’t just start using Kubernetes just because it’s the new shiny thing in tech right now.

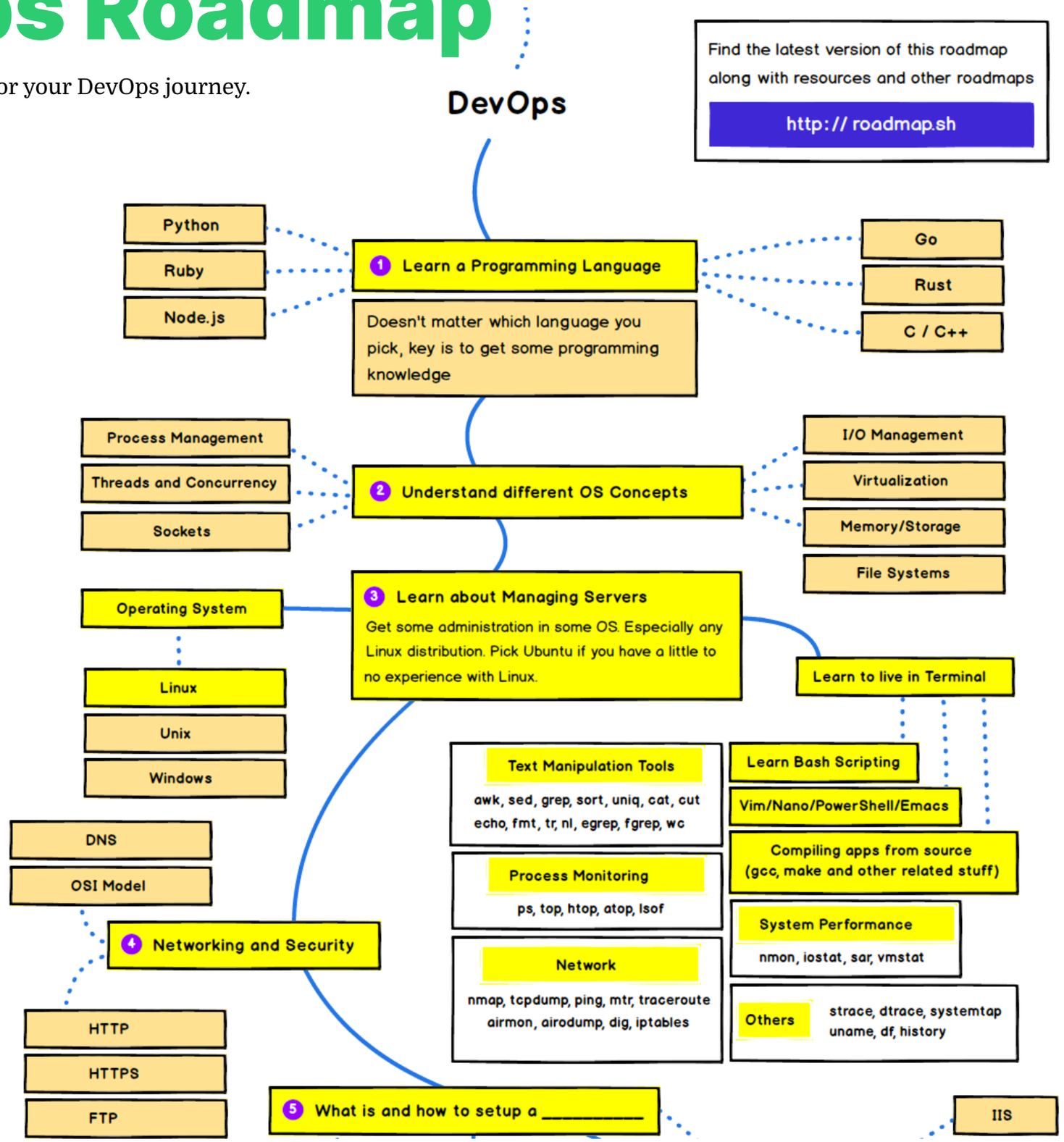
Use it because you know WHY you might need it.

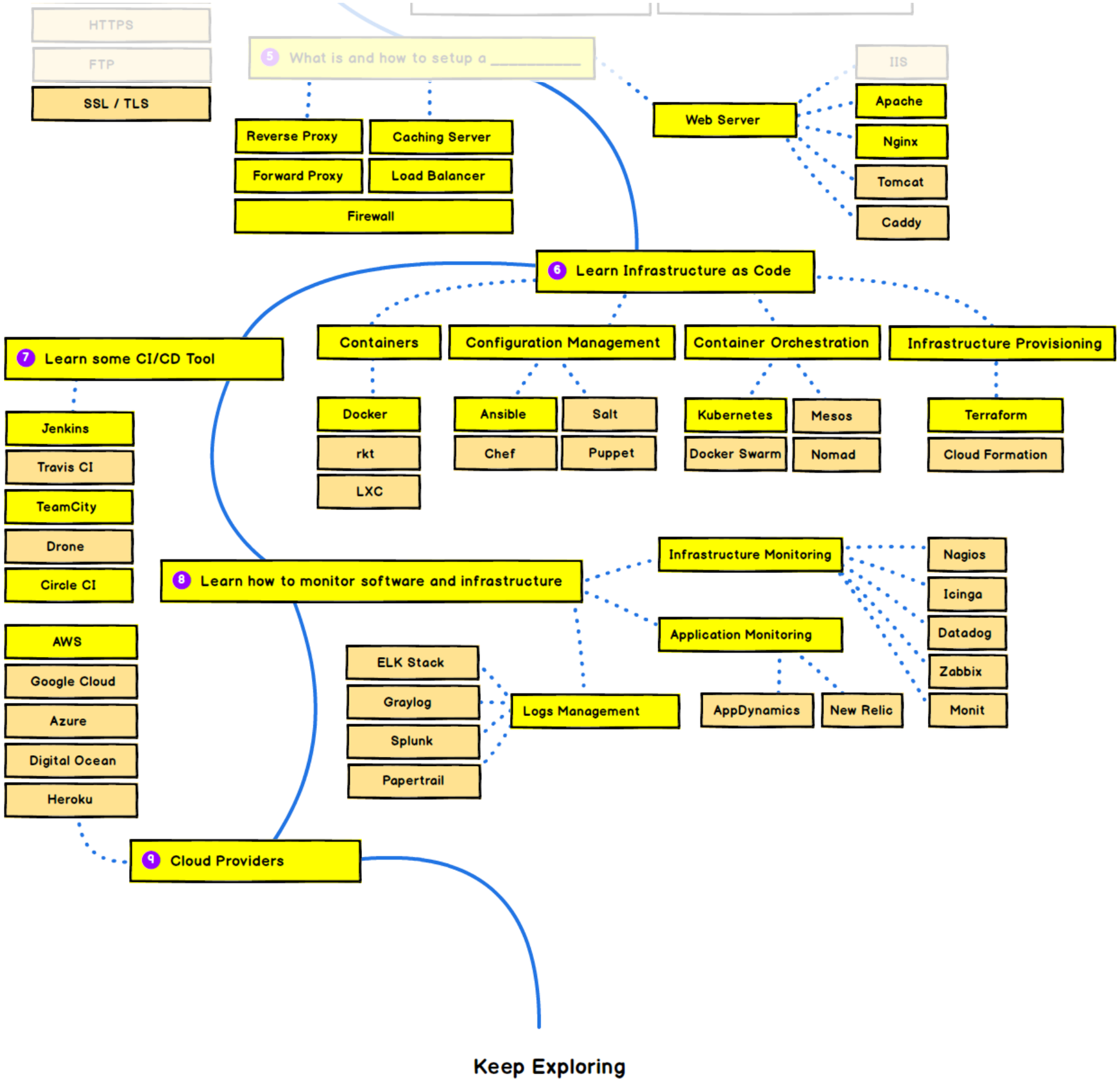
What does a project truly need? That is for you to decide.



DevOps Roadmap

Finally, here's a roadmap for your DevOps journey.




HTTPS
FTP
SSL / TLS
5 What is and how to setup a _____
Reverse Proxy
Caching Server
Forward Proxy
Load Balancer
Firewall
Web Server
IIS
Apache
Nginx
Tomcat
Caddy
6 Learn Infrastructure as Code
Containers
Configuration Management
Container Orchestration
Infrastructure Provisioning
7 Learn some CI/CD Tool
Jenkins
Travis CI
TeamCity
Drone
Circle CI
Docker
rkt
LXC
Ansible
Salt
Chef
Puppet
Kubernetes
Mesos
Docker Swarm
Nomad
Terraform
Cloud Formation
8 Learn how to monitor software and infrastructure
Infrastructure Monitoring
Application Monitoring
Logs Management
Nagios
Icinga
Datadog
Zabbix
Monit
AppDynamics
New Relic
ELK Stack
Graylog
Splunk
Papertrail
AWS
Google Cloud
Azure
Digital Ocean
Heroku
9 Cloud Providers
Keep Exploring

Parting Thoughts

Like everything in our industry, DevOps is an ever-evolving process. Tools and services that you use now might become obsolete in the future. However, that does not mean that the principles you learned cannot be used again. Keep researching existing and new things available in the market and finally, just ask yourself, “Will this make my life any easier?”. If you see the benefits, go ahead. If not, well, on to the next one.

References

AWS: What is DevOps?

https://aws.amazon.com/devops/what-is-devops/

12 Factor App

https://12factor.net/

How to become a DevOps Engineer In Six Months or Less

https://medium.com/@devfire/how-to-become-a-devops-engineer-in-six-months-or-less-366097df7737

DevOps Roadmap

https://roadmap.sh/devops

leapfrog

Leapfrog Technology, Inc is a global full-service technology services company. Founded in 2010, we have offices in Seattle, Boston and Kathmandu. With a high performing team of over 200 engineers, designers, and product analysts, we help companies solve technology innovation and digital transformation challenges.

We have instilled a culture of continuous learning in our people. Our mission is to be the best and most trusted technology services company to execute clients' digital vision.

leapfrog

DevOps Handbook

info@lftechnology.com

www.lftechnology.com