DevOps

Raúl Estrada

Octubre 2020

5. DevOps Practices

There are several practices to follow, but the idea should not use any as a gold standard. We should carefully examine every practice in the background of our state and objectives and then make informed decisions. However, almost all the practices tend to focus on automating processes as much as possible.

5.1. Agile Planning

Agile planning is the practice of defining the work in short increments. While the end objective should be clear, it's not necessary to define and detail the entire application upfront. The key here is **to prioritize work based on the value it can deliver**.

Then, it should be broken in an iteration of short but functioning increments.

5.2. Infrastructure-as-Code (IaC)

This is the **practice of managing and provisioning infrastructure through machine-readable configuration files**. We also manage these configurations in a version control system like we manage our codebase. There are many domain-specific languages available to create these configuration files declaratively.

5.3. Test Automation

Software testing has been traditionally a manual effort often conducted in silos. This does not marry well with agile principles. Hence, it's imperative that we try **to automate software testing at all levels, such as unit testing, functional testing, security testing, and performance testing**.

5.4. Continuous Integration (CI)

Continuous integration is the **practice of merging working code more often in small increments to a shared repository**. Usually, there are automated builds and checks frequently running on this shared repository to alert us of any code breaks as soon as possible.

5.5. Continuous Delivery/Deployment (CD)

Continuous delivery is the **practice of releasing software in small increments as soon as it passes all checks**. This is often practiced together with Continuous Integration and can benefit from an automated release mechanism (referred to as Continuous Deployment).

5.6. Continuous Monitoring

Monitoring – perhaps the center of DevOps – enables faster feedback loops. Identifying **the right metrics to monitor all aspects of the software, including infrastructure, is crucial**. Having the right metrics, coupled with real-time and effective analytics, can help identify and resolve problems faster. Moreover, it feeds directly into the agile planning.

This list is far from complete and is ever-evolving. Teams practicing DevOps are continuously figuring out better ways to achieve their goals. Some of the other practices worth mentioning are Containerization, Cloud-Native Development, and Microservices, to name a few.

6. Tools of the Trade

No discussion on DevOps can be complete without talking about the tools. This is one area where there has been an explosion in the last few years. There may be a new tool out there by the time we finish reading this tutorial! While this is tempting and overwhelming at the same time, it's necessary to exercise caution.

We mustn't start our DevOps journey with tools as the first thing in our minds. We **must explore and establish our goals, people (culture), and practices before finding the right tools**. Being clear on that, let's see what are some of the time-tested tools available to us.

6.2. Development

The idea behind agile is to prototype faster and seek feedback on the actual software. Developers must make changes and merge faster into a shared version of the software. It's even more important for communication between team members to be fluid and fast.

Let's look at some of the ubiquitous tools in this domain.

Git **is a distributed version control system.** It's fairly popular, and there are numerous hosted services providing git repositories and value-added functions. Originally developed by Linus Torvalds, it makes collaboration between software developers quite convenient.

6.3. Integration

Changes merged by developers should be continuously inspected for compliance. What constitutes compliance is specific to team and application. However, it's common to see static and dynamic code analysis, as well as functional and non-functional metric measurements, as components of compliance.

Let's look briefly at a couple of popular integration tools.

Jenkins **is a compelling, open-source, and free automation server.** It has been in the industry for years and has matured enough to service a large spectrum of automation use cases. It offers a declarative way to define an automation routine and a variety of ways to trigger it automatically or manually. Moreover, it has a rich set of plugins that serve several additional features to create powerful automation pipelines.

6.4. Delivery

To deliver changes and new features to software quickly is important. As soon as we've established that the changes merged in the repository comply with our standards and policies, we should be able to deliver it to the end-users fastly. This helps us gather feedback and shape the software better.

There are several tools here that can help us to automate some aspects of delivery to the point where we achieve continuous deployment.

Docker **is a prevalent tool for containerizing any type of application quickly.** It leverages OS-level virtualization to isolate software in packages called containers. Containerization **has an immediate benefit in terms of more reliable software delivery**. Docker Containers talk to each other through well-defined channels. Moreover, this is pretty lightweight compared to other ways of isolation like Virtual Machines.

Chef/Puppet/Ansible **are configuration management tools.** As we know, an actual running instance of a software application is a combination of the codebase build and its configurations. And while the codebase build is often immutable across environments, configurations are not. This is where **we need a configuration management tool to deploy our application with ease and speed**. There are several popular tools in this space, each having their quirks, but Chef, Puppet, and Ansible pretty much cover the bases.

HashiCorp Terraform **can help us with infrastructure provisioning**, which has been a tedious and time-consuming task since the days of private data centers. But with more and more adoption of cloud, infrastructure is often seen as a disposable and repeatable construct. However, this can only be achieved if we've got **a tool with which we can define simple to complex infrastructure declaratively and create it at the click of a button**. It may sound like a dream sequence, but Terraform is actively trying to bridge that gap!

6.5. Monitoring

Finally, to be able to observe the deployment and measure it against the targets is essential. There can be a host of metrics we can collect from systems and applications. These include some of the business metrics that are specific to our application.

The idea here is to be able to collect, curate, store, and analyze these metrics in almost real-time. There are several new products, both open source, and commercial, which are available in this space.

Elastic-Logstash-Kibana (ELK) is a stack of three open-source projects — Elasticsearch, Logstash, and Kibana. Elasticsearch is a highly-scalable search and analytics engine. Logstash provides us a server-side data processing pipeline capable of consuming data from a wide variety of sources. Finally, Kibana helps us visualize this data. Together, this stack can be **used to aggregate data like logs from all the applications and analyze them in real-time**.

Prometheus is an open-source system monitoring and alerting tool originally developed by SoundCloud. It comes with a multi-dimensional data model, a flexible query language, and can pull time-series data over HTTP. **Grafana is another open-source analytics and monitoring solution** that works with several databases. Together, Prometheus and Grafana can **give us a real-time handle on pretty much any metric that our systems are capable of producing**.