DevOps Tools

The best DevOps tools for your application lifecycle

Application lifecycle: Writing code

Source Control

There is no doubt anymore. Everyone is using **Git** for version control of their source code. The basic commands like clone, commit, and push are easy to learn, but there's much more. Can you create a branch? Rebase your branch from master? Resolve conflicts? Squash and merge your branch into master? What if you accidentally made a wrong commit? If you don't know the answer to these questions, you should get yourself up to speed. Any real Site Reliability Engineer **(SRE)** or Cloud / DevOps engineer will be able to use these commands without hesitation.

Software Development Platform

Git integrates with a Software Development Platform like **GitHub**, **GitLab**, **Atlassian**, **Azure DevOps**, or **AWS CodeCommit**. These platforms allow developers to collaborate on software, open Pull Requests **(PRs)**, and resolve conflicts. You want to understand how to use these platforms. These platforms allow you to collaborate on your projects with your colleagues. Engineers building deployment pipelines will need to know how to integrate source control within the Continuous Delivery platform.

Build and test:

Continuous Delivery

In this stage, you typically use your code within version control to create pipelines to build, test and deploy your software. Software Development Platforms often provide tooling within their platform to build pipelines. If that's the case and you're starting from the ground up, then most likely you'll be using the built-in pipeline functionality. It'll be fully integrated within the platform, using integrations for source control, testing (build tool integrations), and deployment (for example, **Kubernetes**).

Besides using your Software Development Platform, a popular open-source tool that has already been around for some time is **Jenkins**. It is used in almost all organizations. **Jenkins** can be used as a complete Continuous Delivery platform, taking care of the build, test and deployment phases. It can also be integrated within existing platforms. **AWS CodeBuild**, for example, can build & test your software itself, but you also have the option to integrate with **Jenkins** and let Jenkins handle the build, test, and deploy phase.

Deployment:

Continuous Deployment

As part of the Continuous Delivery cycle, you need to be able to Continuously Deploy your applications. Today you typically deploy using containers, so let's cover this first.

To deploy containers, you'll need a container orchestration platform. The most popular one is **Kubernetes**. It is available as a hosted platform on almost all public cloud computing providers. Although **Kubernetes** can do a lot for you, it is a very complex tool with its own ecosystem of tools. If you are looking for something more simple, then have a look at what cloud vendors are offering. **AWS** provides Elastic Container Service **(ECS)**. **ECS** is also a container orchestrator, but much simpler to use. Other useful tools are Docker-compose to build and test your containers locally, and Docker swarm, which is a Docker orchestrator built by Docker itself.

Operation:

Building infrastructure

Before you can deploy on your (cloud) infrastructure, you still need to set up a lot of resources in the cloud (or on-premises). The most common tool that can help you with that is **Terraform**. Terraform supports the major cloud providers (Amazon AWS, Microsoft Azure, Google Cloud, and others). It lets you write your Infrastructure as Code, allowing you to abstract your complete infrastructure setup. You can store the code in version control, allowing you to collaborate with teammates, get a history of changes, and use auditing tools.

Monitoring and security:

Security Threats

Building all the infrastructure to execute your **DevOps** strategy shouldn't be done without security in mind. Cloud Providers have a lot of tools in place to help you secure your data and lower the risk of data breaches. One very powerful and often overlooked measure is simply to use very tight access rules. **AWS** has Identity & Access Management **(IAM)** to create users, groups, and roles. These contain access rules, which are called policies. Too often those policies are written too broadly, allowing too much access for a user, group or role. Tightening them up more could significantly improve your security posture. Advanced policies can be written with specific conditions to test where access is originating from, and allowing or denying access to resources based on this information.

Monitoring Tools

Once your infrastructure is set up, you'll need to monitor it. The monitoring will be 2-fold: Monitoring of applications, and monitoring of the infrastructure itself.

To monitor the infrastructure, you're going to want to use the tools that your cloud provider provides you. For **AWS** this will be **CloudWatch**. If no monitoring is available or you're not on the cloud, then you'll definitely want to have a look at Prometheus. This monitoring tool integrates with most cloud-native tooling and is a great tool to use. You will find a lot of different agents that you can use. You can install those agents on your Linux / Windows server instances. Other plugins are available for specific services, like databases. Prometheus supports pulling the metrics, but also has a push gateway for parts of your system that can't support pull.