Ifeakachukwu Owoh

Understanding of a DevOps culture in an Organization.

'DevOps' and DevOps culture by extension is something that has evolved to have no specific meaning and will differ from one organization to the next. A startup with little to no users will have a different DevOps methodology from a team building an application for enterprise users or a heavy traffic social media application. In general, it can refer to the mindset that a company has towards increasing the efficiency, security and the speed of software development, delivery and the complete Software Development life cycle compared to traditional methods. It has to do with how a company embraces methodology and tooling that can help design and deliver software better and facilitate automation to give the business a competitive advantage. The way a company goes about streamlining the communication between traditional Development and Operation teams and even subsets of the engineering team is vital. At every point in the Software Development life cycle there are things that can be carried out to ensure that work is done in the most automated and efficient way possible. There are a few out of the many pillars of DevOps engineering in any company that I will highlight.

- Pull Request Automation: This includes anything from automated code review & scanning, notification to reviewers, per change ephemeral environments, Continuous integration.

- Deployment Automation: Continuous Deployments, type of deployment strategies(rolling vs canary), deployment rollbacks(using source control).

- Application Performance Management: Metrics, Logging, Monitoring, Alerting.

In general it is most ideal for a company to adopt DevOps practices and tooling based on what is appropriate for their stage, scale and what is preferred by their developers. For a company that has no users, painstakingly implementing continuous delivery to kubernetes might be overkill; but it could be just right for companies that need the scale and functionality that a tool like kubernetes will offer.

What does a DevOps Engineer do? and what tools are available.

A DevOps Engineer's job is to introduce and use these before-mentioned tools, methodologies and processes to make optimizations throughout the software development lifecycle for increased efficiency and automation. It can range from anything between writing code for application development, to server maintenance and administration. Understanding that traditionally, Development teams want to build new features fast, and operations teams want to preserve stability and reliability of an application, it is up to a DevOps engineer to implement practices that make both things possible. Their goal is to automate as much of the Software design and delivery process as possible. At Discord, a DevOps engineer once described a task she embarked on after scrutinizing their continuous integration workflow, which was to write a proprietary sorting algorithm to sort pull requests that typically sit in a large pool by order of their priority. While this may not be considered by most as conventional DevOps work, this is the mindset that a good DevOps engineer should hold, and the job should be to automate, secure and streamline the development process with whatever languages and tools are available to you. That said, there a few things that constitute conventional DevOps work, and I will highlight a few of them and the tools used for them:

- Continuous Integration/Continuous Deployment:
  Tools like Jenkins, Github Actions, GCP Cloudbuild and the likes are used to create CI/CD pipelines to automate code deployment workflow.

- Infrastructure as Code: A DevOps engineer uses tools like Terraform, Ansible and Cloudformation to define infrastructure with code to take advantage of a few benefits like state management, version control, and proga[ramming language functionality. It helps automate the manual creation and configuration processes for infrastructure.

- Scripting: Languages like Bash and Python allow DevOps engineers to write scripts that can automate tedious tasks.

- Logging, Monitoring and Metrics: A DevOps engineer should use tools for logging and monitoring the tech infrastructure that can help speed up the development process and be aware of application issues. Tools like Prometheus, Grafana, Datadog, and Uptime Kuma are used.

Describe the Software Development Lifecycle identifying where DevOps fits in.

Software Development Life Cycle outlines all the stages and steps required in building a software application. What it looks like can differ from one organization to the next, but in general there are 6 steps in the process;

- Plan Stage: Stage meant for defining the vision, gathering requirements, sharing tasks, and planning the development of the application. Tools used are Jira, Trello, Linear etc.

- Design Stage: Engineers analyze requirements and identify best solutions to develop the software. Tools used are Figma, Notion etc

- Development Stage: Writing and managing code, setting up development environments and ensuring code quality. Tools used are Git, Programming languages, Text Editors eg Vim.

- Testing Stage: This stage involves ensuring code quality through automated tests and continuous integration. Tools used are: Selenium, Jenkins etc.

- Deployment Stage: This involves automating the release process to ensure code is released quickly and reliably. Tools used are Jenkins, Ansible, Kubernetes.

- Maintenance: This involves Operations and Monitoring; Continuously managing, operating, and monitoring applications and Infrastructure to ensure performance and reliability Tools used are Prometheus, Grafana, Datadog.

DevOps Best Practices

AUTOMATE EVERYTHING:

Automation is at the heart of DevOps. From code builds to tests and other things, automating these tasks can help us focus on the things that require attention and reduce the risk of human error and miscommunication.

IMPLEMENT CONTINUOUS INTEGRATION AND DELIVERY(CI/CD):

Set up a robust CI/CD pipeline that includes automated tests, integration steps and deployment stages to catch issues early and deploy updates swiftly. This way, the repository is always in a good state since we introduce small changes that are easy to handle.

ADOPT A MICROSERVICES ARCHITECTURE:

While Microservices aren't a catch-all for software development and may not be ideal for every single situation, creating modular and loosely coupled applications help create smaller, independent services that can be developed, deployed and scaled independently.

CREATE A CULTURE OF COMMUNICATION AND COLLABORATION:

Devops is understandably as much about culture as it is about tools and processes. Breaking down silos and emphasizing communication allows people to collaborate freely and removes the fear of failure. Use tools like slack and teams to promote open communication in a DevOps setting.

IMPLEMENT A DEPLOYMENT STRATEGY:

Deployment strategies help to minimize downtime and reduce the risk of introducing new changes by gradually rolling out updates. For instance, Implement blue-green deployments to switch traffic between two environments, and use canary deployments to release updates to a small subset of users before a full rollout.

BUILD FOR SCALE:

Prioritize scalability and performance in DevOps when building products; ensuring your application can handle increased load and perform well under stress is crucial for maintaining a good application.

DevOps vs SRE

DevOps and SRE(Site Reliability Engineering) are both approaches aimed at improving efficiency and reliability of software applications and operations but the primary difference is focus. DevOps focuses on the end to end lifecycle and bridging the gap between Development and operations teams and automating workflows to speed up software delivery. SRE, developed at Google, applies Software Engineering principles to operations with core emphasis on delivery and the stability of the production environment. Both can be implemented hand in hand in an organization, and in some cases SRE is a subset of DevOps.

SRE uses metrics like Service Level Objectives or Indicators to track error rates against expected results. While DevOps aims to streamline the entire software lifecycle, SRE zeroes in on keeping services running smoothly in production, often employing techniques like error budgets and automated incident response. Both approaches are complementary, with DevOps focusing on collaboration and speed, and SRE emphasizing reliability and operational excellence. DevOps uses metrics like Deployment Frequency and Mean time to Recovery.

For a team of engineers, the main difference between DevOps and SRE will come down to their focus and daily tasks. DevOps engineers work on breaking down barriers between development and operations to streamline the software delivery process. They're all about automating CI/CD pipelines and ensuring smooth, fast deployments.

On the other hand, SREs focus on keeping services reliable and performing well in production. They use engineering methods to tackle operational problems, automate infrastructure management, and handle incidents. SREs set reliability goals and use error budgets to balance new features with system stability.