DevOps

What is DevOps ? 🪶

📜DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams.

📜DevOps is the next evolution of agile methodologies.

📜The term DevOps, a combination of the words development and operations, reflects the process of integrating these disciplines into one, continuous process.

📜It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

📜The DevOps movement began around 2007 when the software development and IT operations communities raised concerns about the traditional software development model, where developers who wrote code worked apart from operations who deployed and supported the code.

📜A DevOps team includes developers and IT operations working collaboratively throughout the product lifecycle, in order to increase the speed and quality of software deployment.

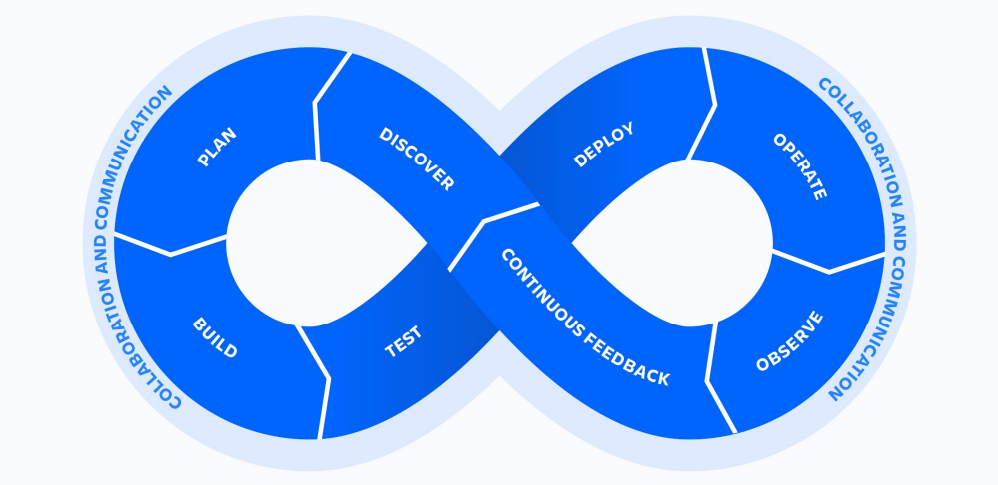
The DevOps lifecycle .. 🪶

📜The DevOps lifecycle consists of eight phases representing the processes, capabilities, and tools needed for development (on the left side of the loop) and operations (on the right side of the loop).

📜It outlines the journey of a software application from initial development to continuous delivery and ongoing operations.

📜Throughout each phase, teams collaborate and communicate to maintain alignment, velocity, and quality.

DevOps Lifecycle Phases .. 🚀



DevOps tools .. 🚀

📜When it comes to a DevOps toolchain, organizations should look for tools that

1-improve collaboration

2-reduce context-switching

3- introduce automation and leverage observability

4-and monitoring to ship better software, faster.

The DevOps toolchain comprises a set of tools and technologies that facilitate the automation, integration, and management of the various processes and stages within the DevOps lifecycle.

There are two primary approaches to a DevOps toolchain:

1) **an all-in-one .**

   An all-in-one DevOps solution: provides a complete solution that usually doesn't integrate with other third-party tools.

2) **open toolchain.**

   An open toolchain: can be customized for a team's needs with different tools.

   Regardless of the type of DevOps toolchain , a DevOps process needs to use the right tools to address the key phases of DevOps Lifecycle .

   DevOps tools can be categorized: into different groups based on the stages of the DevOps lifecycle they support .

The selected tools touch multiple phases of the DevOps lifecycle.

Discover :

In the Discover phase, a DevOps team researches and defines the scope of a project. In particular, it involves activities such as user research, establishing goals, and defining success.

Building software is a team sport. In preparation for the upcoming sprint, teams must workshop to explore, organize, and prioritize ideas. Ideas must align to strategic goals and deliver customer impact. Agile can help guide DevOps teams.

Tools like Mural and Miro empower the entire software team to gather ideas and conduct research. Jira Product Discovery organizes this information into actionable inputs and prioritizes actions for development teams. As you're prioritizing, you'll also need to keep your backlog of user feedback in mind.
Product discovery is the very first activity of a product design, which then becomes the baseline for decision making. During product discovery, you can collect all the crucial information about any user problems and then provide solutions for them.

We recommend looking for tools that encourage “asynchronous brainstorming”. It’s important that everyone can share and comment on anything: ideas, strategies, goals, requirements, roadmaps, and documentation.

Plan :

Look for tools that provide sprint planning, issue tracking, and allow collaboration, such as Jira. DevOps teams should adopt agile practices to improve speed and quality.

Agile is an iterative approach to project management and software development that helps teams break work into smaller pieces to deliver incremental value.

Another great practice is continuously gathering user feedback, organizing it into actionable inputs, and prioritizing those actions for your development teams. Look for tools that encourage “asynchronous brainstorming” (if you will). It’s important that everyone can share and comment on anything: ideas, strategies, goals, requirements, roadmaps and documentation.And don’t forget about integrations and feature flags.

Build :

Production-identical environments for development

Git is a free and open source version control system. It offers excellent support for branching, merging, and rewriting repository history, which has led to many innovative and powerful workflows and tools for the development build process.

While Puppet and Chef primarily benefit operations, developers use open source tools like Kubernetes and Docker to provision individual development environments. Coding against virtual, disposable replicas of production helps you get more work done.

Infrastructure as code

Infrastructure as code means re-provisioning is faster than repairing – and more consistent and reproducible. It also means you can easily spin up variations of your development environment with similar configuration as production. Provisioning code can be applied and reapplied to put a server into a known baseline. It can be stored in version control. It can be tested, incorporated into CI (continuous integration), and peer-reviewed.

Source control and collaborative coding

It's important to have source control of your code. Source control tools help store the code in different chains so you can see every change and collaborate more easily by sharing those changes. Rather than waiting on change approval boards before deploying to production, you can improve code quality and throughput with peer reviews done via pull requests.

Source control tools should integrate with other tools, which allows you to connect the different parts of code development and delivery. This allows you to know if the feature's code is running in production. If an incident occurs, the code can be retrieved to shed light on the incident.

Continuous feedback :

Continuous integration is the practice of checking in code to a shared repository several times a day, and testing it each time. That way, you automatically detect problems early, fix them when they're easiest to fix, and roll out new features to your users as early as possible.

DevOps teams should evaluate each release and generate reports to improve future releases. By gathering continuous feedback, teams can improve their processes and incorporate customer feedback to improve the next release.

Code review by pull-requests requires branching and is all the rage. The DevOps North Star is a workflow that results in fewer and faster branches and maintains testing rigor without sacrificing development speed.

Pull requests notify team of changes for review before integration, improving software quality and development speed.

Test :

Automated testing

Continuous integration (CI) allows multiple developers to contribute to a single shared repository. When code changes are merged, automated tests are run to ensure correctness before integration. Merging and testing code often help development teams gain reassurance in the quality and predictability of code once deployed.

Testing tools span many needs and capabilities, including exploratory testing, test management, and orchestration. However, for the DevOps toolchain, automation is an essential function. Automated testing pays off over time by speeding up your development and testing cycles in the long run. And in a DevOps environment, it's important for another reason: awareness.

Test automation improves software quality, reduces risk, and provides reports and graphs to identify risky areas.

Deploy :

Deployment dashboards

Continuous deployment (CD) allows teams to release features frequently into production in an automated fashion. Teams also have the option to deploy with feature flags, delivering new code to users steadily and methodically rather than all at once. This approach improves velocity, productivity, and sustainability of software development teams.

One of the most stressful parts of shipping software is getting all the change, test, and deployment information for an upcoming release into one place. The last thing anyone needs before a release is a long meeting to report on status. This is where release dashboards come in.

Automated deployment

No universal solution for automated deployment. Use Ruby or bash scripts. Use variables to factor out host names. Avoid duplicated code. Peer review scripts.

Automate deployments starting from lowest-level environment to production. Highlights environment differences and generates tasks for standardization. Reduces "server drift" with automation.

Operate :

Incident, change and problem tracking

Manage the end-to-end delivery of IT services to customers. This includes the practices involved in design, implementation, configuration, deployment, and maintenance of all IT infrastructure that supports an organization's services.

The keys to unlocking collaboration between DevOps teams is making sure they're viewing the same work. What happens when incidents are reported? Are they linked and traceable to software problems? When changes are made, are they linked to releases?

Nothing blocks Dev's collaboration with Ops more than having incidents and software development projects tracked in different systems. Look for tools that keep incidents, changes, problems, and software projects on one platform so you can identify and fix problems faster.

Observe :

Application and server performance monitoring

There are two types of monitoring that should be automated: server monitoring and application performance monitoring.

Quickly identify and resolve issues that impact product uptime, speed, and functionality. Automatically notify your team of changes, high-risk actions, or failures, so you can keep services on.

Continuous observability is essential for successful DevOps teams to understand trends and monitor the overall health of their applications and environments. Manual spot-checking is insufficient for this purpose, and software that can continuously listen and record data is necessary.

Continuous Feedback :

Continuous feedback is crucial for understanding whether a product meets customer needs. This feedback can be collected through various channels such as NPS data, churn surveys, bug reports, support tickets, and even social media. In a DevOps culture, everyone on the product team has access to user comments, which helps inform decision-making processes.

Integrate chat tools with survey platforms and social media for real-time feedback. Investing in a social media management platform can provide deeper insights and

historical data reports. Analyzing and incorporating feedback is more efficient in the long run than releasing unwanted features.