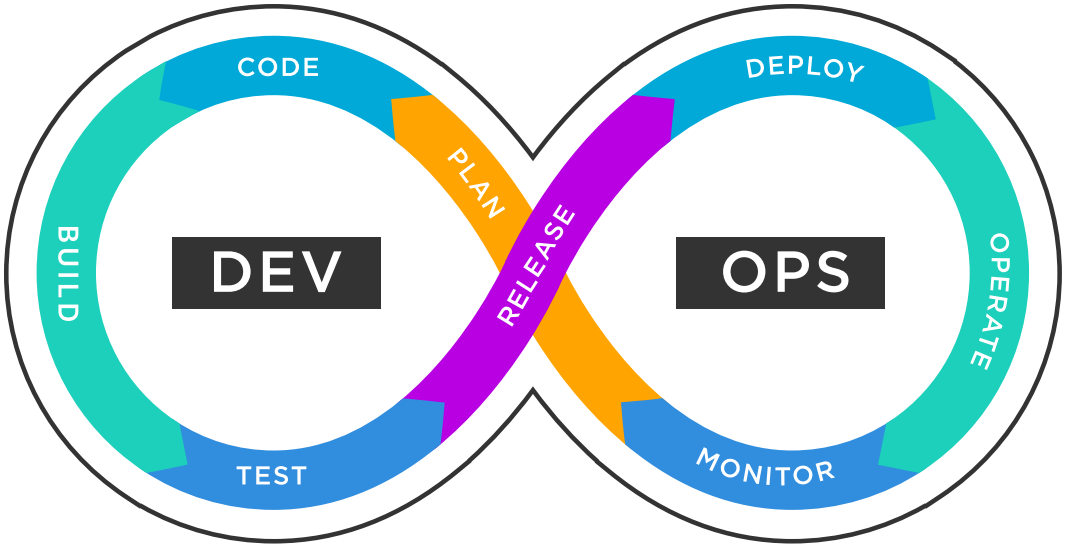
DEVOPS BOOK

PETRO KOLOSOV

Contents

1. DevOps definition and Cycle

DevOps – is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality [1].



(1) **Plan**

The first stage is the planning of our future product or functionality. At this stage, we need to answer three simple questions:

- **Who are we doing this for?**

  By asking this question, we need to determine who our ultimate target audience is – the audience that will use our service.

- **What are we doing?**

  By asking this question, we must precisely define the technical specifications for our future product.

- **Why are we doing it this way?**

  By asking this question, we need to determine why certain architectural or other decisions were made, and why specific components were chosen. Each decision and its justification should be documented.

(2) **Code**

At this step, developers write, manage, and maintain code collaboratively using version control systems. The focus is on creating high-quality, maintainable, and secure code.

(3) **Build**

At this step, the source code is compiled, tested, and packaged into deployable artifacts. This phase ensures the software is functional, free from major bugs, and ready for deployment to staging or production environments.

(4) **Test**

At this step, the assembled application is tested on environment that closely mirrors production. This involves launching the application and verifying its functionality as well as the functionality of the entire system. In addition to basic testing, integrations with other software components are also checked for proper operation.

(5) **Release**

This step is informational in nature. The notification to the responsible administrator includes information that a new release has been prepared, it has passed all necessary tests, and is ready for deployment on the production platform.

(6) **Deploy**

At this step, release artifact is being deployed to production environment. It involves releasing and distributing the application, making it accessible to end-users. This phase ensures that the deployment is seamless, consistent, and efficient.

(7) **Operate**

The step where application management takes place. This includes its configuration, granting access to end users, and managing response to incidents and problems.

(8) **Monitor**

At this step, we collect information about the application's performance: its metrics, logs, and user feedback. After that, we move back to the Plan stage again. Our infinity has come full circle.

2. Possible tasks per Cycle

Plan

- Define business and technical requirements for each sprint
- Align with stakeholders on goals, capacity planning, and risk assessment
- Document architectural decisions and system design updates
- Create and groom backlog items based on monitoring and feedback loops

Code

- Configure GIT repositories branch protection rules, access for developers
- Choose convenient branching strategy for effective development
- Configure pull request validation pipelines for new code
- Perform code review of new code
- Enforce code style and formatting rules via linters and pre-commit hooks
- Document code properly (e.g., README, usage examples, API docs)
- Ensure license and dependency tracking for open-source usage

Build

- Configure build pipelines and trigger policies based on main branch
- Implement unit and integration tests as part of build CI/CD pipeline
- Configure security scans like Sonarcloud to ensure build artifact is secure
- Implement efficient multi-stage build process for time saving
- Implement automatic artifact versioning according to SemVer
- Performance boost of build pipelines using parallel jobs and caching
- Use build matrices for testing across multiple platforms/versions
- Integrate static code analysis tools beyond Sonarcloud if needed

Test

- Provision and configure testing environments
- Configure deployment pipelines for testing environments

- Resolve configuration issues on testing environments
- Monitor and resolve issues raised by QA team
- Configure proper VPN access for developers
- Define test strategy: unit, integration, load, smoke, e2e, and security tests
- Automate test report generation and publication
- Perform automated test coverage reporting
- Configure test data management strategies

Release

- Designed and documented release process
- Designed and documented hotfix strategy for production environment
- Worked closely with managers and QA team to prepare release
- Automate release notes generation
- Include rollback planning and test coverage before release approval

Deploy

- Configure production environment using Terraform
- Implement efficient release pipelines to production
- Configure secure access to production environment
- Troubleshoot and fix the issues in release pipelines
- Configure production environment using Ansible
- Build and push docker images to container registry
- Implement secure access to secrets during deployments
- Implement canary or blue-green deployments strategy
- Define deployment validation steps (automated smoke tests)
- Ensure compliance and audit logging post-deploy

Operate

- Perform smoke test and fixing configuration issues after deployment
- Configure users and developers access to deployed applications

- Response to incidents after production deployments
- Perform regular system patching and maintenance tasks
- Document and automate standard operating procedures (SOPs)
- Cost optimization and resource governance (e.g., Azure Advisor usage)

Monitor

- Set up dashboards using Prometheus, Grafana, Azure Monitor, etc.
- Configure alerts for infrastructure and application health
- Define and monitor SLIs, SLOs, and error budgets
- Log aggregation and centralization (e.g., ELK stack, Azure Log Analytics)
- Analyze post-mortems and incident trends

Security (Cross-Cutting)

- Integrate dependency scanning tools (e.g., Snyk, OWASP Dependency Check)
- Perform secret scanning and management (e.g., GitLeaks, Azure Key Vault validation)
- Enforce Role-Based Access Control (RBAC) across environments

Documentation & Collaboration (Cross-Cutting)

- Maintain centralized wiki or knowledge base for onboarding and SOPs
- Regular DevOps retrospectives and continuous improvement sessions

3. Technologies per DevOps Cycle

(1) **Plan**

Tools: JIRA, Confluence, Azure DevOps (Boards), Azure Portal, Azure CLI

(2) **Code**

Tools: GIT, GitHub Actions, Bitbucket, Azure DevOps (Repos), C#, PowerShell, .NET Core, Chocolatey, SonarCloud, MendBolt, Snyk

(3) **Build**

Tools: Azure Pipelines, GitHub Actions, YAML, CI/CD, SemVer, .NET Core, SonarCloud, MendBolt, Snyk, Docker

(4) **Test**

Tools: Azure Pipelines, PowerShell, Terraform (for test infra), .NET Core, Docker, YAML, Azure DevOps (Pipelines), Test frameworks (xUnit), Packer, VPN, Azure App Service, Azure VM

(5) **Release**

Tools: Azure DevOps (Release pipelines), Azure Pipelines, Release management, JIRA, Confluence, YAML, SemVer

(6) **Deploy**

Tools: Terraform, Ansible, HELM, Docker, Azure Pipelines, Azure DevOps (Pipelines), Azure App Service, Azure VM, Azure KeyVault, Azure Automation, SSH, Chocolatey, RBAC, Kubectl, K9s

(7) **Operate**

Tools: Windows Server, Linux Server, Azure VM, Azure Portal, Azure CLI, IIS, Nginx, Azure App Service, Azure Automation, PowerShell, SSH, RBAC, LetsEncrypt, Cloudflare

(8) **Monitor**

Tools: Prometheus, Grafana, Azure Monitor, Azure DevOps (Dashboards), CURL, Telnet, K9s, Azure CLI

References

[1] Len Bass, Ingo Weber, and Liming Zhu. What Software Architects Need to Know About DevOps. 2015. https://www.informit.com/articles/article.aspx?p=2424801.

`0.1.42+main.4b7fbe6`