Mindtree
A Larsen & Toubro Group Company

GitLab-based DevOps Platform

__GitLab__ is one of the most popular DevOps platforms, and provides an ecosystem for code and release management and the pipeline tools for end-to-end management. In this whitepaper, we discuss the methods and best practices to implement DevOps using the GitLab ecosystem, and cover its tools, branching strategy and design.

Welcome to possible

A Mindtree Whitepaper

GitLab-based DevOps

GitLab Tools

GitLab is a full-fledged DevOps system that provides a software repository, CI/CD system, user management and monitoring from a single ecosystem as a Software as a Service (SaaS) offering with industry standard end-to-end security.

Apart from traditional command line tools, it provides webUI for both operations, maintenance and performing regular branching, tagging, merging and any other Git operations. These rich user interfaces are available for CI/CD pipeline monitoring, status check and configurations as well.

GitLab enables high flexibility to integrate most other third party tools such as SonarQube, build scripts for the most modern technology stacks and automation, as well as deployment plugins to cloud labs such as Firebase Cloudlab and BrowserStack. Apart from regular environment-based deployment in cloud hosting, it also supports multi-cloud based deployment.

The GitLab DevOps ecosystem ensures continuous feedback, which is a key need for agile development. We have depicted the GitLab ecosystem in Figure 1.

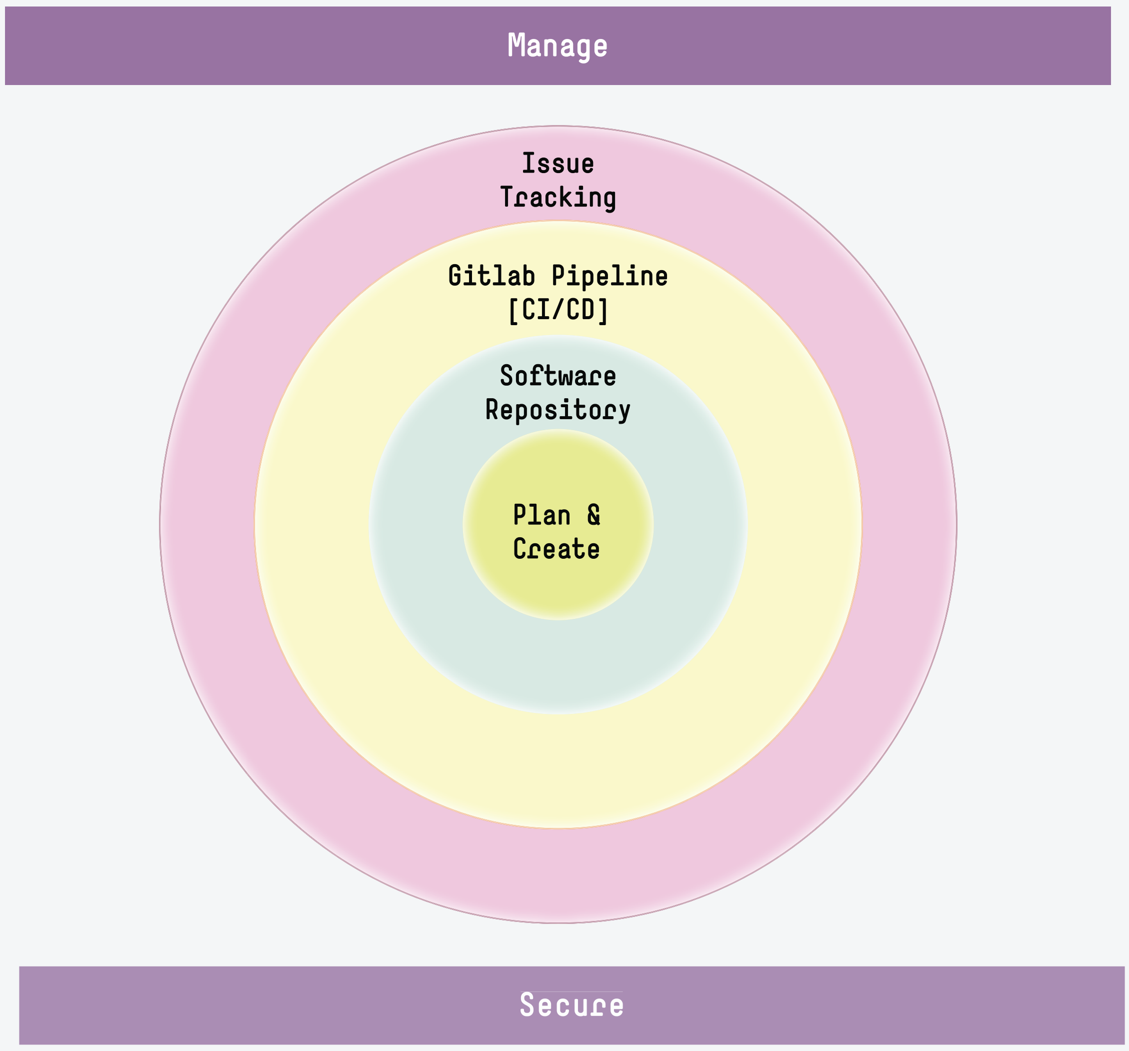

Figure 1 GitLab ecosystem overview

Plan & Create

As part of plan and create program, the manager plans the release and defines milestones. The business analysts author user stories and collaborate with team members.

Software Repository

A version-controlled software code management system that is easy to create and manage with workflows for each solution tenant in the architecture. The repository's access is supported with industry standard security like user management with role-based access management, multi factor authentications and SSL handshakes. This code base can be configured with workflows and branching strategies, while code review stages can be defined by integration to internal or external tools.

GitLab Pipeline[CI/CD]

The crux of the solution support for a whole devops ecosystem is provided via the GitLab pipeline, where a continuous integration and continuous deployment is managed. Modern solutions such as a docker container-based scalable DevOps build pipelines are available with a flexibility of integrating a private runner or an on-premises / local build system. Multi-cloud-based solution deployment is supported with ease, while enough plugins and tools are available that will help assess the code quality. Additionally, automated test cases can be executed, while release management is supported via build deployment to environments or external distribution entities such as appcenter, testflight, Play Store and app stores.

Issue Tracking

Monitoring and issue tracking is supported, which helps tag/manage issues. The release supports continuous delivery in addition to continuous integration and deployment. It also has an automated issue tracking system that can help issue reporting at each stage of continuous development and deployment, which enables seamless tracking for continuous delivery that meets the defined quality standards.

Manage and Secure

As part of manage, we handle various phases of the program, and as part of security, we manage the overall security of the solution.

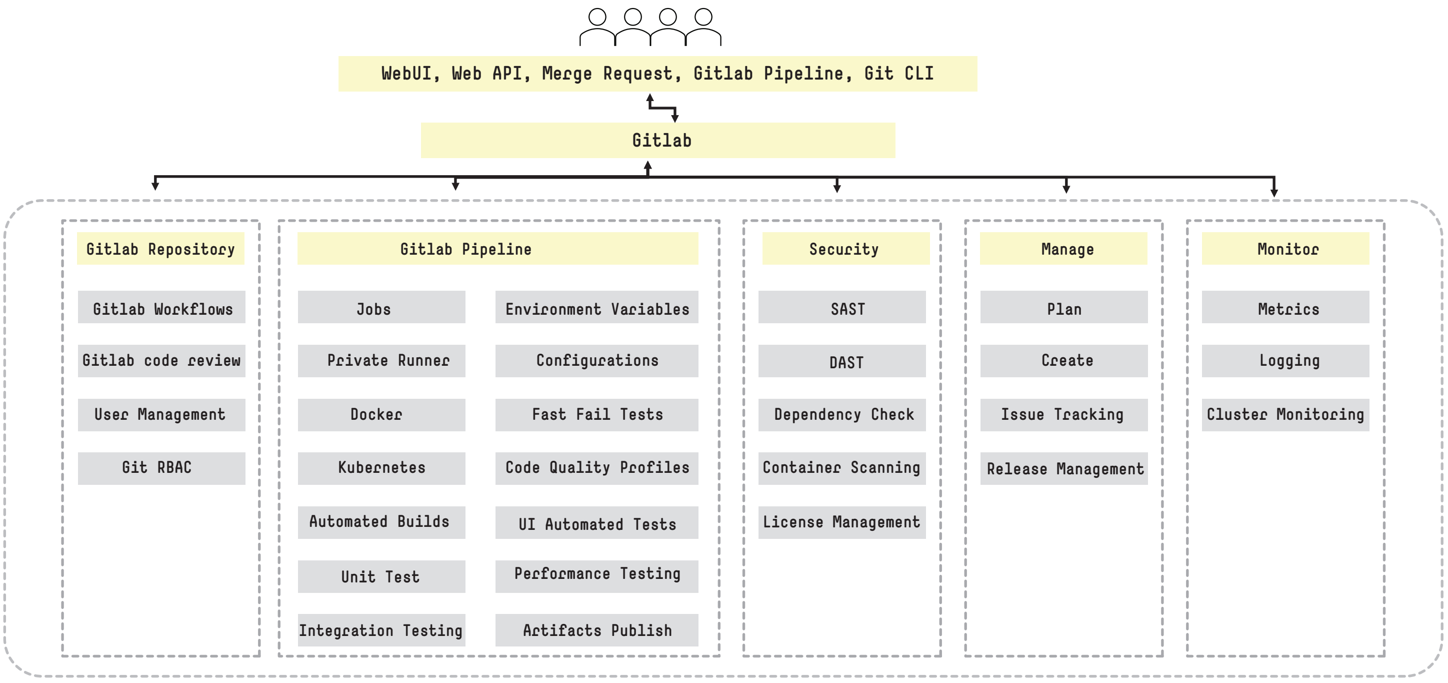

Figure 2: Overview of GitLab tools and interfaces for end users

We have depicted various GitLab tools across various solution pillars needed for a full-fledged DevOps implementation. For each solution pillar, GitLab has various corresponding tools that help meet the devops needs. The Kubernetes and docker-based solution is purely the next gen solution for a Devops platform.
It has a sophisticated mechanism for stakeholders and users such as project management, development team, quality assurance team, devops engineers and business users, enabled with a rich user interface and visual dashboards for metrics. This enables effective collaboration from all participating teams in all roles and responsibilities.
For the compliance needs, Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), license management, dependency check and container scanning tools take care of quality standards, and automated checks makes continuous delivery adhere to quality profiles.
Let us check the GitLab devops setup and configuration process to follow for dependencies that need to be identified for a program that has both cloud and mobile apps deployment.

Given below are the Prerequisites for GitLab devops setup,

1. Enterprise subscription
2. Creation of a software repository
3. Repo workflow
4. Branching strategy
5. Environment strategy
6. GitLab pipeline strategy
7. Tools consideration in each GitLab pipeline stages
8. Quality profiles and gates for each stage or overall pipeline
9. Build deployment and distribution strategy
10. App promotion strategy

Tools and strategy consideration for GitLab DevOps

Design

Given below are the main design considerations:

- Single eco system for end-to-end devops
- Managed SaaS-based solution for DevOps
- Easy to configure and scalable DevOps for a large-scale program with a multi-pod-based diverse technology stack
- Industry standard security, manage and monitor mechanism
- Continuous delivery that suits the agile methodology-based project

Drivers

- Git way of working for developers, devops and release managers
- Optimized infrastructure maintenance
- Steps and stage-wise configuration with automated issue tracking, quality gates for standard and security scans for compliance

Motivations

- Automated devops system that lets everyone in the team to contribute, brings in collaboration, enables remote monitoring and performs automated issue reporting and tracking for a continuous delivery
- Keeps system as well as source code secured and compliant with security standards and quality gates

  We have depicted a sample GitLab based DevOps in Figure 3

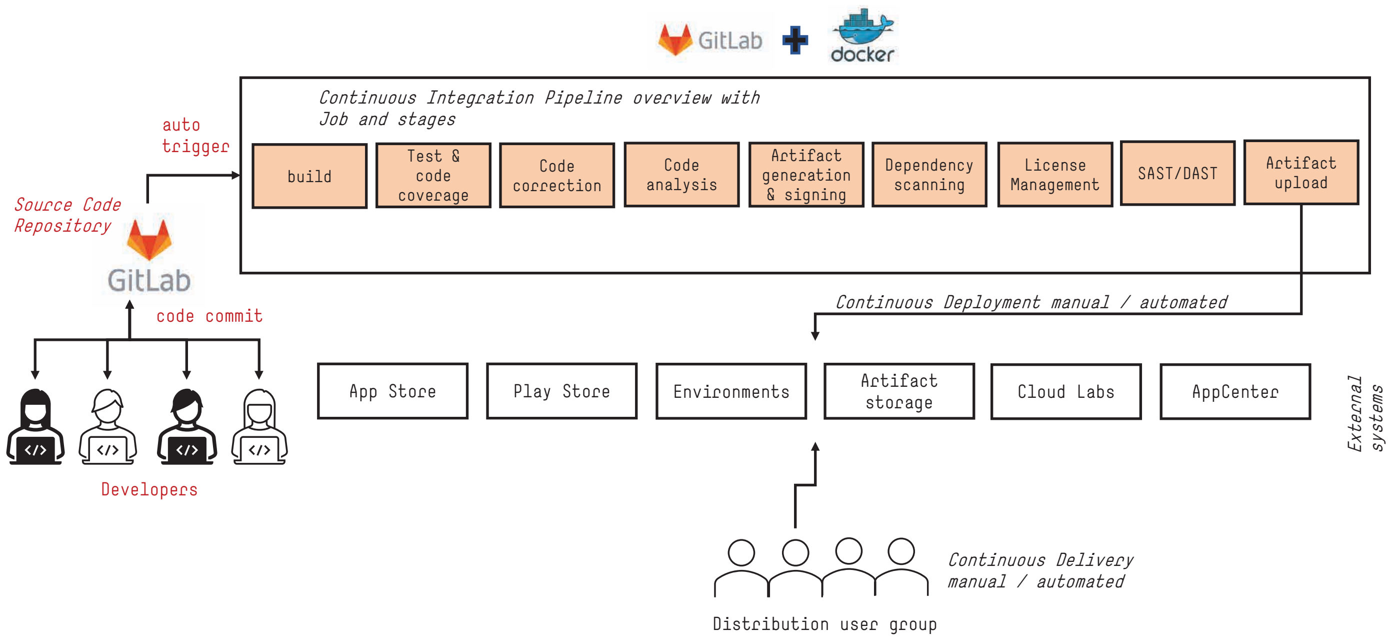

Figure 3: GitLab DevOps overview with ci/cd pipeline job and stages

Branching Strategy

Designing and implementing appropriate branches is crucial to implement an extensible development and deployment strategy. In this section, we have laid out a few GitHub-based branching best practices.

To start with, we need to define the core branches based on the development and deployment needs. In Figure 4, we have defined the branching strategy that we commonly use.

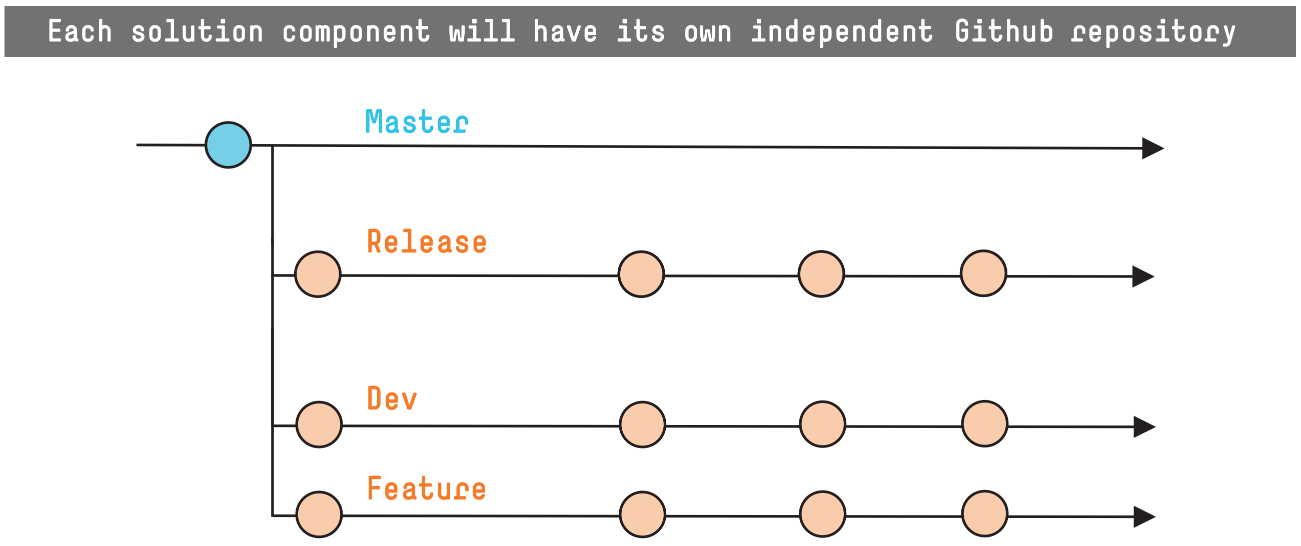
Each solution component will have its own independent Github repository
Master
Release
Dev
Feature

Figure 4 Key Branches

The main branches are as follows:

- Feature branch is used for user story/feature development
- Dev branch is used for Dev environment
- Release branch is used for stable releases
- Master branch represents what's in production currently or is production-ready.

In Figure 5, we have depicted the process of a release across all these branches.

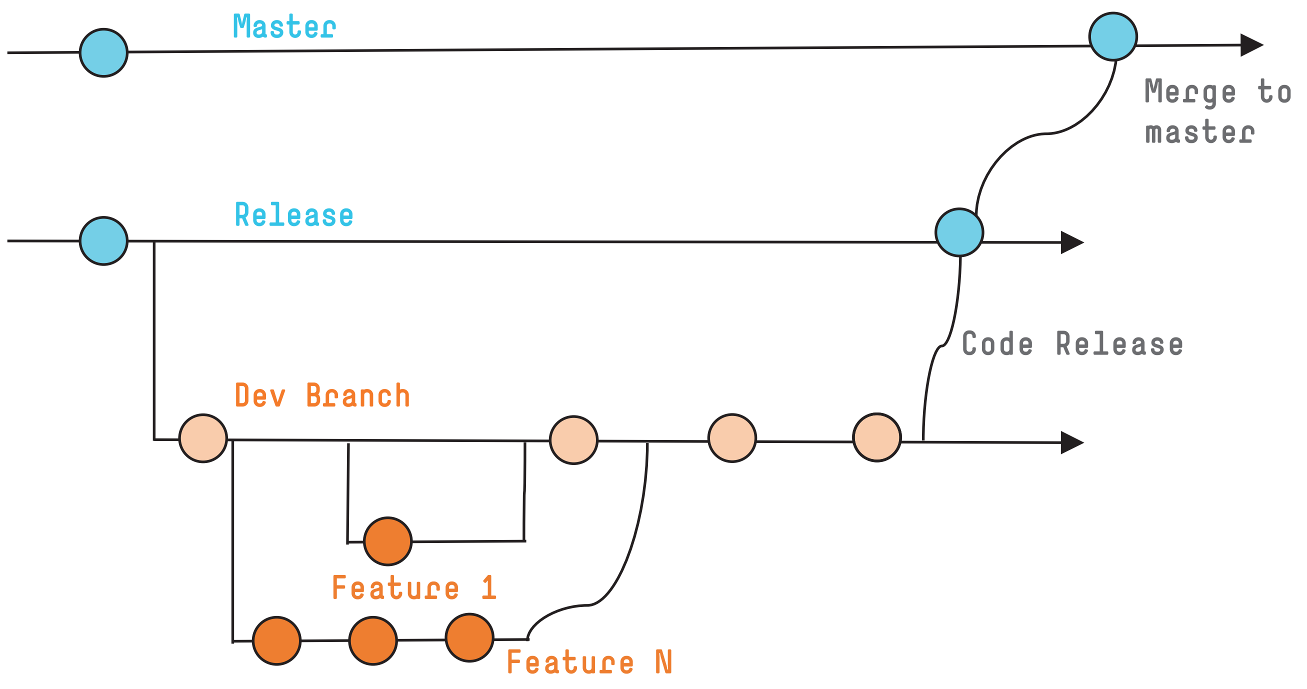

Figure 5 Release Flow

Given below are the high level steps in the release management flow:

1. User stories are developed in separate Feature branches and merged back into the Dev branch
2. Dev branch allows bug-fixing, hardening and other release preparation to continue in isolation. No feature development should happen on these branches.
3. Dev branch will be protected so that code in feature branches can be reviewed and approved by designated reviewers before merging into release branch. This can be implemented using built-in Pull request and protection features of GitHub.
4. QA-approved build version gets deployed to UAT and other pre-production environments. If there are any defects, it will be done from the Dev branch.
5. Once the code is stable enough for release, it will be pushed to the Release branch. The code in this branch is tagged with the release number.
6. The code from Release branch will be pushed to the Master branch for production deployment.

At a high level, the flow will be like this:

$$\begin{tikzcd}\text{Feature Branch} \arrow[r] & \text{Dev Branch} \arrow[r] & \text{Release Branch} \arrow[r] & \text{Master Branch}\end{tikzcd}$$

To handle the production bug fixes, we use a hotfix branch as depicted in Figure 6. A branch will be created as needed for hotfixes in production. These will originate from the master branch and merged back into master as well as the dev branch for inclusion in future releases. Hotfix branches should be very short-lived.

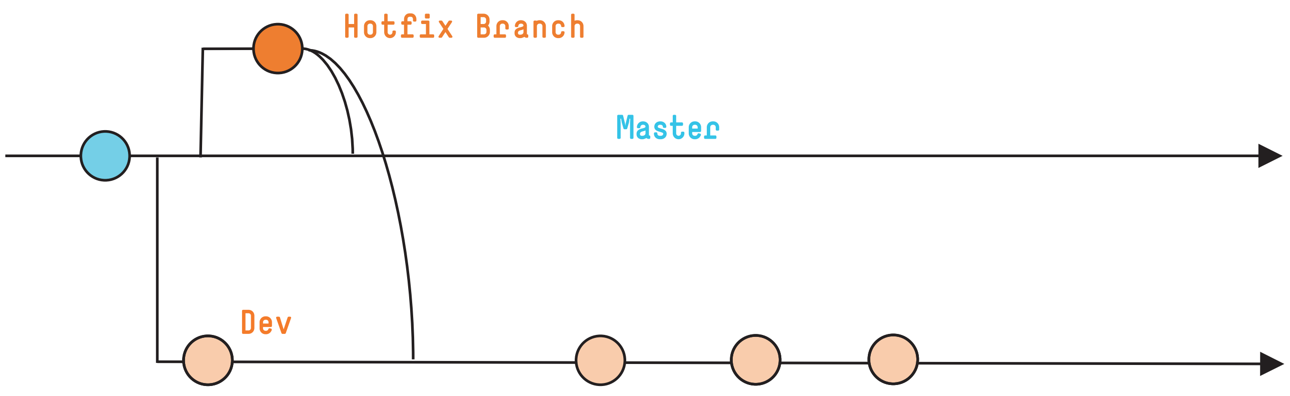

Figure 6 Hotfix branch

DevOps Implementation

The start of the pipeline is based on the code commit, which can either be automated or set for manual execution. Each of these start a stage that can either have a single or multiple jobs with an objective that decides if the stage has been successfully completed. Quality gates can be defined for each stage to manage the quality levels.

Build

It is a build phase to generate the binary files in either debug or release mode. The GitLab pipeline .yml file will have the package manager dependency such as Maven, NPM, Pega, to compile the binary using gradle, npm etc.

Test and Code coverage

Unit test cases and fast fail test cases can be configured here to identify the code quality. Additionally, UI automation test cases can also be executed in this phase. The end objective of this phase is to identify the code coverage in terms of percentage through these test cases.

Code correction

Code analyzer and lint tools perform code correction checks and report identified issues in this stage.

Code analysis

SonarScanner or SonarQube-based code analysis is performed in this stage. This stage can also be configured to consolidated issues identified by the code correction phase, so that all issues are available from a single SonarQube dashboard.Quality gate rules can be defined at this stage before a final artifact is generated such that only if the rules are passed, the pipeline proceeds to the next stage.

Artifact generation and signing

Artifact generation is a phase where a final distributable binary is created that also includes signing the artifact to make it secure and ensure that code obfuscation is achieved.

Dependency scanning

Dependency scanning helps determine if the external code integrated in the binary such as plugins, sdks, frameworks, library contains any vulnerabilities.

License management

As part of this phase, we attribute all the licensed software used in the application along with their licenses.

Artifact Upload

At this stage of a pipeline, the artifacts upload to perform a deployment to external systems such as App store / Play store for mobile apps or multi-cloud-based environment for specific deployment of web apps so that the backend solution can be enabled. A successful deployment means a successful pass of all quality gates set for previous stages of pipeline, thus assuring compliance to security and quality standards.

Tools and strategy followed in setting up GitLab based DevOps:

Category	Sample Tools
Source code repository	GitLab
Code Review	GitLab
Code Merge and Commit	GitLab, SourceTree, Android Studio, GitLab client plugins for IDE
Continuous Integration	GitLab Pipeline
CI Infrastructure	Docker + Private runner for iOS
Technology Stack	Swift for iOS, Kotlin for Android, Angular
Dependency Manager	CocoaPods, Maven, NPM
Build Tools	XCode, Gradle, NPM
Monitoring	New Relic
Code Analysis	Cloud-hosted SonarQube, Swift Lint, SonarLint, Android Kotlin, Android Lint, Codelyzer, TSLint
Unit Test	XCTest, Junit, Fragment Scenarios, ngtest
Code Coverage	Jacoco, XCTest, SonarQube, ngtest
Version Increment	Auto increment on patch version Manual configuration for major.minor
Alerts and notification	On build failure with changelist On successful upload of build artifacts
Security Tool (SAST/DAST)	OWASP Dependency Check, Burb Suit, App Scan
Artifact Upload	AppCenter, JFrog, AWS EC2, TestFlight, Play Store, Browser Stack
App Promotion	DEV -> QA -> FVT->FIT->SIT->Staging->UAT->PreProd->Prod Each upgrade is based on QA and business user sign off

Best Practices

1. Docker-based hosting reduces the infrastructure maintained and the build time
2. Private runner for iOS compilation is better than cloud-based solutions for cost effectiveness
3. Automated issue tracking to monitor the health of code commit

Advantages of GitLab

- For a whole DevOps setup, configure and effective utilization, and realization for an agile development project, it is a one stop solution that enables effective collaboration. It lets everyone in the team contribute, and provides easy remote monitoring, both visual and automated alerts, including pipeline workflows
- GitLab can scale up to large enterprise programs effectively. It can also be customized to small scale industries and startups to suit their business needs
- From an infrastructure point of view, it lets users choose and integrate their choice of infrastructure via a private runner or a sophisticated SaaS offering over Docker
- GitLab provides multiple deployment provisions including multi-cloud deployment, which are the latest needs for modern enterprise applications and end-to-end solutions till front end channels such as Web application, mobile apps and chatbots.
- Since GitLab is an opensource platform, its features and functionalities are constantly evolving, addressing the needs of modern applications
- Since GitLab is a SaaS offering, reliable uptime is achieved for development cycles and meets the velocity for delivery. Concurrent execution of GitLab runner associated with a job enables faster build pipeline execution. From the whole pipeline perspective, there can be multiple jobs that are executed at the same time.

About the Authors

Sandhya B

Sandhya is a Senior Architect who has worked on multiple mobile projects with the in-house devops setup and configuration of these systems. With hands-on experience of using in-house setup and identifying real benefits of streamlined release management, she has greatly helped in delivering quality apps to customers.

Dr. Shailesh Kumar Shivakumar

Dr. Shailesh Kumar Shivakumar is a Solution Architect and has 19+ years of experience in a wide spectrum of digital technologies including, enterprise portals, content management systems, lean portals, and microservices. Dr. Shailesh holds a PhD degree in computer science and has authored eight technical books published by the world's top academic publishers such as Elsevier Science, Taylor and Franscis, Wiley/IEEE Press, and Apress. Dr. Shailesh has authored more than 14 technical white papers, five blogs, twelve textbook chapters for various under-graduate and post graduate programs and has contributed multiple articles. He has published 20+ research papers in reputed international journals. Dr. Shailesh holds two granted US patents, apart from ten patent applications. Dr. Shailesh has presented multiple research papers at international conferences. Dr. Shailesh's Google Knowledge Graph can be accessed at https://g.co/kgs/4YoaiN . He has successfully led several large scale digital engagements for Fortune 500 clients. Shailesh can be reached at Shaileshkumar.Shivakumarasetty@mindtree.com

About Mindtree

Mindtree [NSE: MINDTREE] is a global technology consulting and services company, helping enterprises marry scale with agility to achieve competitive advantage. "Born digital," in 1999 and now a Larsen & Toubro Group Company, Mindtree applies its deep domain knowledge to 275+ enterprise client engagements to break down silos, make sense of digital complexity and bring new initiatives to market faster. We enable IT to move at the speed of business, leveraging emerging technologies and the efficiencies of Continuous Delivery to spur business innovation. Operating in more than 15 countries across the world, we're consistently regarded as one of the best places to work, embodied every day by our winning culture made up of over 22,000 entrepreneurial, collaborative and dedicated "Mindtree Minds."

www.mindtree.com