DevOps

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.
www.fandsindia.com

Ground Rules

- Turn off cell phone. If you cannot please keep it on silent mode. You can go out and attend your call.
- If you have any questions or issues please let me know immediately.
- Let us be punctual.

Agenda

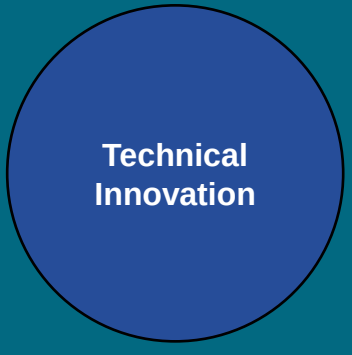
Discuss

Why Change?

fands™
A Fast AND Steady Approach

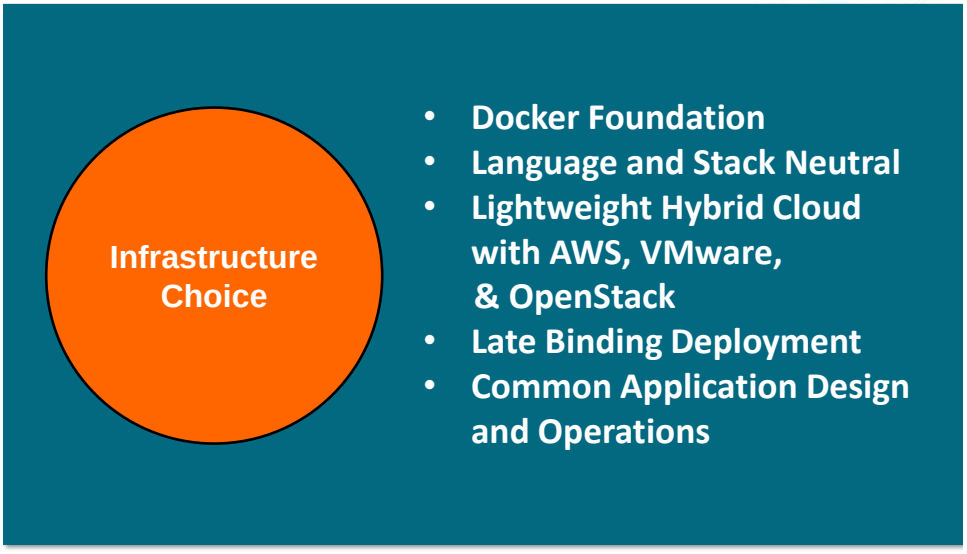
- Time-to-Market Acceleration
- Experimentation
- Rapid Prototyping
- Flexible Partnering
- IoT/IoS Support

fands™
A Fast AND Steady Approach

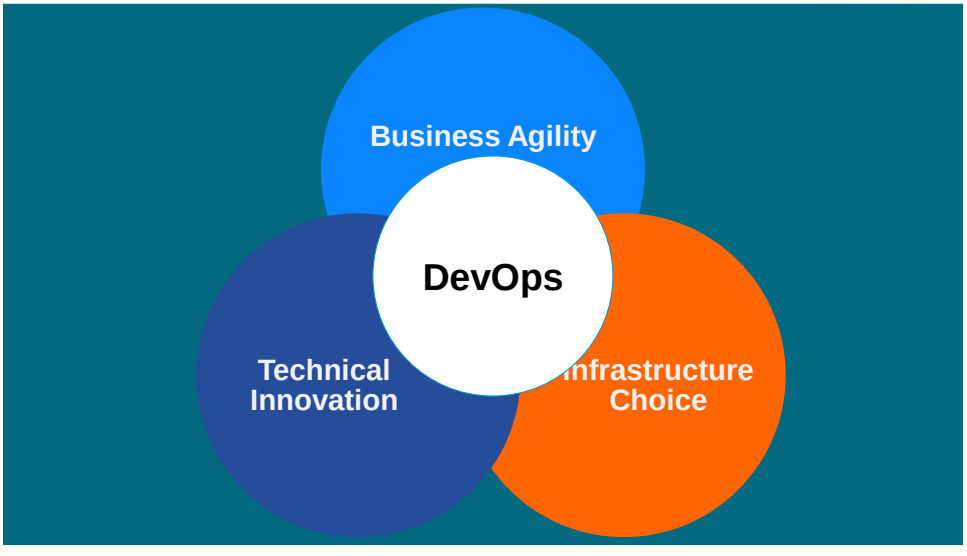


- Polyglot Enablement
- DevOps Automation
- API Support
- Microservices Architecture
- Blue-Green Deployment
- Application Scaling and Elasticity
- PaaS

fands™
A Fast AND Steady Approach



fands™
A Fast AND Steady Approach



Monoliths to Micro Services

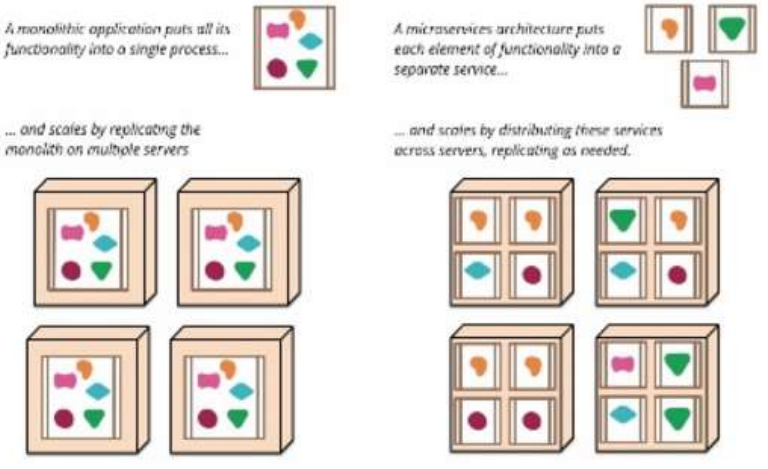
Monoliths to Micro Services

- Monoliths
  - single deployment
  - single runtime
  - single codebase
  - interaction between classes is most often synchronous
  - most often every layer is in a separate package

Monoliths to Micro Services

- Microservices
  - many small modules with specific functionality
  - more than one codebase
  - every microservice is a separate deployment
  - every microservices has its own DB
  - communication with web- services
  - ensures module independence

Monoliths Vs Micro Services

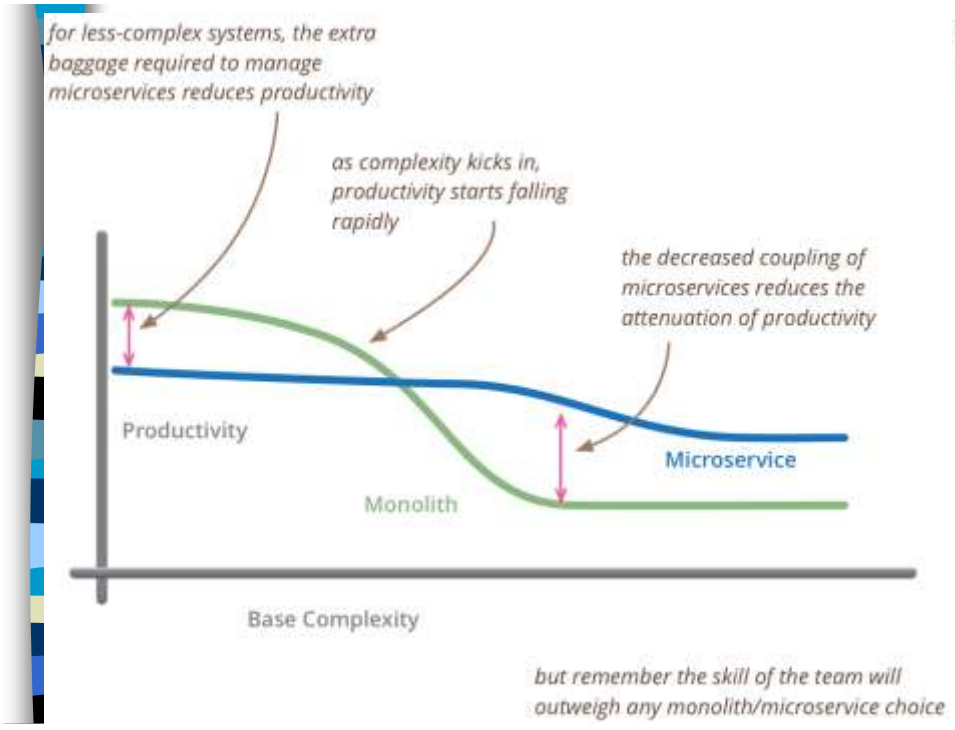


Monoliths Vs Micro Services

- Downsides of monoliths…?
  - "spaghetti" / "big ball of mud"
  - you need a full redeploy
  - programmers often violate the layer boundaries
  - classes often "leak" their implementation
  - hard to work with multiple teams
  - hard to manage

Monoliths Vs Micro Services

- Benefits of Microservices..?
  - modelled around the business domain
  - deployment automation culture
  - hides implementation details
  - decentralization
  - option to use multiple languages
  - separate deployment
  - separate monitoring
  - isolating problems

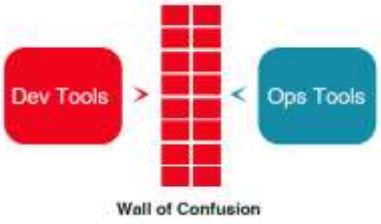


fauds

Issues with Micro Services

- Network overhead
- Transaction coordination
- Need for duplicating common data
  - keeping it in sync
- Complicated deployment pipeline dependencies

Dev-Ops



One Team, One Goal

Dev Ops Focuses both the Apps team's drive for agillity responsiveness and the NOC's concern with quality and stability on the ultimate goal of providing business value

DevOps Is Fixing It

- DevOps is a software development method that highlights collaboration and open communication between teams.
- DevOps teams are composed of developers and operations professionals working together to create sounder, more fail-proof software.
- Teams that have adopted the DevOps ethos have a better handle on their IT incidents and suffer less downtime.

Why DevOps?

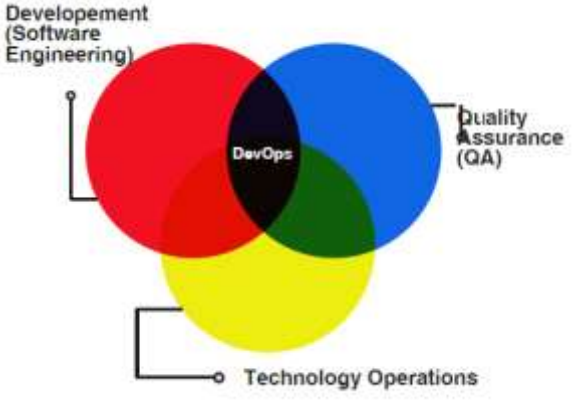
- Never Miss Alerts
  - 31% of DevOps teams said they never miss critical alerts. No other teams could make that claim.
- Respond Faster
  - 75% of DevOps companies said they respond within a half hour. DevOps teams never take longer than an hour to respond.
- Make Your Stakeholders Happy
  - Only 6% of DevOps shops' business stakeholders report dissatisfaction in incident response, versus 30% for non-DevOps teams.
- Keep Your Customers Happy, Too
  - DevOps teams are 30% more likely to be transparent with customers about critical incidents.

DevOps

DevOps is a way of thinking.

CALMS Model

DevOps



Five Basic Principles of DevOps

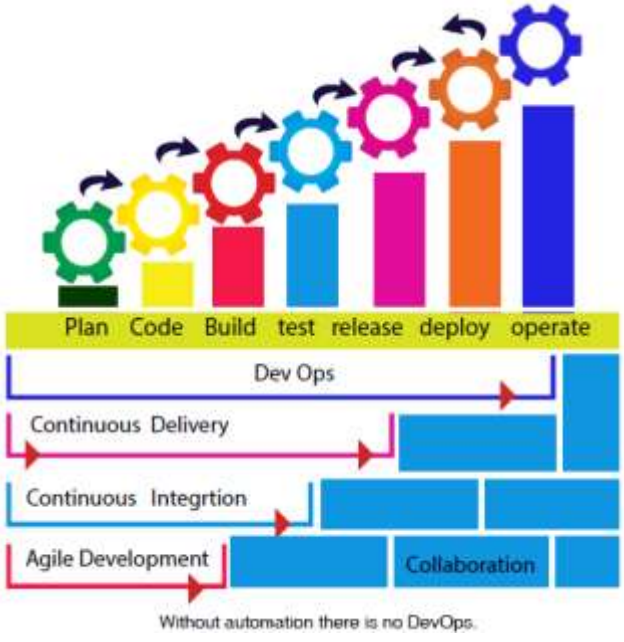
- Eliminate the blame game, Open post-mortems, Feedback, Rewarding failures
- Continous Delivery, Monitoring, Configuration Management
- Business value for end user
- Performance Metrics, Logs, Business goals Metrics,
- People Integration Metrics, KPI
- Ideas, Plans, Goals, Metrics, Complications, Tools

DevOps Combines

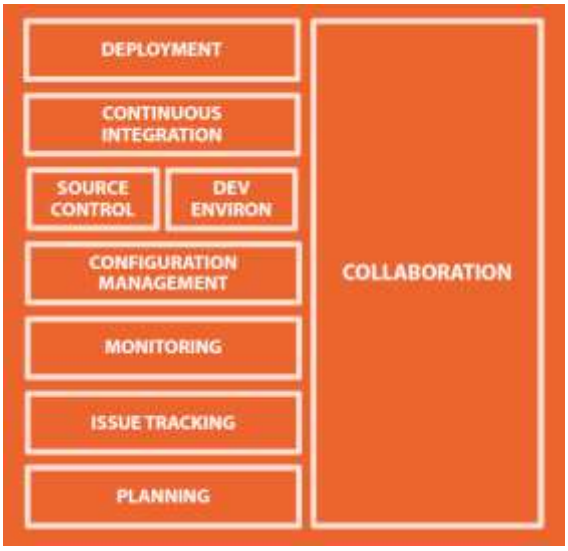
- Develops and verifies against production-like systems
- Reduces cost/time to deliver - Deploy often, deploy faster with repeatable, reliable process
- Increases Quality - Automated testing, Reduce cost/time to test
- Reduces Defect cycle time - Increase the ability to reproduce and fix defects
- Increases Virtualize Environments utilization
- Reduces Deployment related downtime
- Minimizes rollbacks

7Cs OF DevOps

- Communication
- Collaboration
- Controlled Process
- Continuous Integration
- Continuous Deployment
- Continuous Testing
- Continuous Monitoring



DevOps Technology Categories



Collaboration

- Easily Connect Teams
- Tools
  - Skype
  - Slack
  - WordPress
  - GitHub Wiki

Planning

- Shared Vision
- Tools
  - Trello
  - Visual Studio Online

Issue Tracking

- Rapid Response
- Tools
  - ZENDESK
  - Jira
  - Redmine

Monitoring

- Intelligent Co-relation
- Tools
  - Logstash
  - Microsoft System Center
  - Kibana
  - New Relic
  - …..

Configuration Management

- Consistent State
- Tools
  - Chef
  - Salt
  - Puppet
  - Ansible
  - …

Source Control

- Controlled Assets
- Tools
  - GitHub
  - …

Development Environment

- Modern Consistency
- Tools
  - Codenvy
  - Vagrant
  - ..

Continuous Integration

- Incremental Progress
- Tools
  - Jenkins
  - TeamCity
  - Travis CI
  - Go CD
  - Bamboo
  - GitLab
  - Codeship.

Continuous Deployment

- Automated Efficiency
- Tools
  - CloudFormation
  - Packer
  - Docker
  - Octopus
  - Go
  - GitLab

Continuous…


Continuous Development
Continuous Integration
Continuous Testing
Continuous Deployment
Continuous Monitoring
Continuous Feedback
Continuous Delivery

What is Continuous Delivery?

- Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.
- Our goal is to make deployments—whether of a large-scale distributed system, a complex production environment, an embedded system, routine affair that can be performed on demand.
- We achieve all this by ensuring our code is always in a deployable state, even in the face of teams of thousands of developers making changes on a daily basis. We thus completely eliminate the integration, testing and hardening phases that traditionally followed “dev complete”, as well as code freezes.

Continuous Delivery is a small build cycle with short sprints…

- Where the aim is to keep the code in a deployable state at any given time. This does not mean the code or project is 100% complete, but the feature sets that are available are vetted, tested, debugged and ready to deploy, although you may not deploy at that moment.
- *May be our preferred method of working.*

Continuous Deployment

- With **Continuous Deployment**, every change that is made is automatically deployed to production. This approach *works well in enterprise environments where you plan to use the user as the actual tester* and it can be quicker to release.

Continuous Integration

- Continuous Integration is merging all code from all developers to one central branch of the repo many times a day trying to avoid conflicts in the code in the future. The concept here is to have *multiple devs on a project to keep the main branch of the repo to the most current form of the source code, so each dev can check out or pull from the latest code to avoid conflicts.*

DevOps and Agile

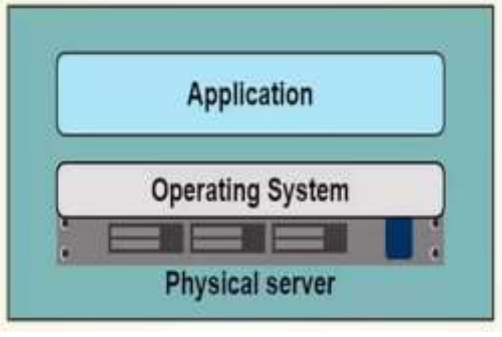
Parameter	Agile	DevOps
What is it?	Agile refers to an iterative approach which focuses on collaboration, customer feedback, and small, rapid releases.	DevOps is considered a practice of bringing development and operations teams together.
Purpose	Agile helps to manage complex projects.	DevOps central concept is to manage end-to-end engineering processes.
Team size	Small Team is at the core of Agile. As smaller is the team, the fewer people on it, the faster they can move.	Relatively larger team size as it involves all the stack holders.
Duration	Agile development is managed in units of "sprints." This time is much less than a month for each sprint.	DevOps strives for deadlines and benchmarks with major releases. The ideal goal is to deliver code to production DAILY or every few hours.
Feedback	Feedback is given by the customer.	Feedback comes from the internal team.
Target Areas	Software Development	End-to-end business solution and fast delivery.

Virtualization

Virtualization

- Process of running a virtual instance of a computer system in a layer abstracted from the actual hardware. Most commonly, it refers to running multiple operating systems on a computer system simultaneously.
- To the applications running on top of the virtualized machine, it can appear as if they are on their own dedicated machine, where the operating system, libraries, and other programs are unique to the guest virtualized system and unconnected to the host operating system which sits below it.

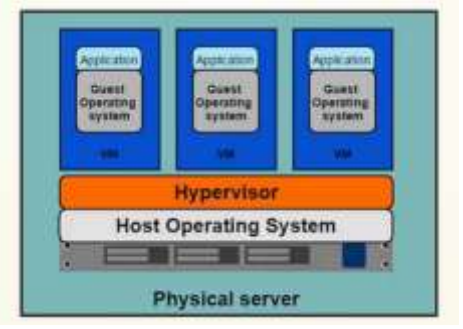
Basic Application Hosting



- Problems
  - Slow deployment times
  - Huge costs
  - Wasted resources
  - Difficult to scale or migrate
  - Vendor lock in

Hypervisor-based Virtualization

- A hypervisor, also known as a virtual machine monitor or VMM, is software that creates and runs virtual machines (VMs).
- A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.



Hypervisor Types

Classification	Characteristics and Description
Type 1: native or bare metal	Native hypervisors are software systems that run directly on the host's hardware to control the hardware, and to monitor the guest operating systems. Consequently, the guest operating system runs on a separate level above the hypervisor. Examples of this classic implementation of virtual machine architecture are Oracle VM, Microsoft Hyper-V, VMWare ESX and Xen.
Type 2: hosted	Hosted hypervisors are designed to run within a traditional operating system. In other words, a hosted hypervisor adds a distinct software layer on top of the host operating system, and the guest operating system becomes a third software level above the hardware. A well-known example of a hosted hypervisor is Oracle VM VirtualBox. Others include VMWare Server and Workstation, Microsoft Virtual PC, KVM, QEMU and Parallels.

Demo

- Oracle VM VirtualBox

Build and Static Code Analysis Tools

Maven Background

- Is a Java build tool
  - "project management and comprehension tool"
- An Apache Project
  - Mostly sponsored by Sonatype
- History
  - Maven 1 (2003)
    - Very Ugly
    - Used in Stack 1
  - Maven 2 (2005)
    - Complete rewrite
    - Not backwards Compatible
    - Used in Stack 2.0,2.1,2.2,3.0
  - Maven 3 (2010)
    - Same as Maven 2 but more stable
    - Used in Stack 2.3, 3.1

Maven Features

- Dependency System
- Multi-module builds
- Consistent project structure
- Consistent build model
- Plugin oriented
- Project generated sites

The Maven Mindset

- All build systems are essentially the same:
  - Compile Source code
  - Copy Resource
  - Compile and Run Tests
  - Package Project
  - Deploy Project
  - Cleanup
- Describe the project and configure the build
  - You don't script a build
  - Maven has no concept of a condition
  - Plugins are configured

Other Java Build Tools

- Ant (2000)
  - Granddaddy of Java Build Tools
  - Scripting in XML
  - Very flexible
- Ant+Ivy (2004)
  - Ant but with Dependency Management
- Gradle (2008)
  - Attempt to combine Maven structure with Groovy Scripting
  - Easily extensible
  - Immature

Maven Learning Resources

- Maven Homepage
  - http://maven.apache.org
    - Reference Documentation for Maven
    - Reference Documentation for core Plugins
- Sonatype Resources
  - http://www.sonatype.com/resource-center.html
    - Free Books
    - Videos

Maven POM

- Stands for Project Object Model
- Describes a project
  - Name and Version
  - Artifact Type
  - Source Code Locations
  - Dependencies
  - Plugins
  - Profiles (Alternate build configurations)
- Uses XML by Default
  - Not the way Ant uses XML

Project Name (GAV)

- Maven uniquely identifies a project using:
  - **g**roupID: Arbitrary project grouping identifier (no spaces or colons)
    - Usually loosely based on Java package
  - **a**rtfiactId: Arbitrary name of project (no spaces or colons)
  - **v**ersion: Version of project
    - Format {Major}.{Minor}.{Maintanence}
    - Add '-SNAPSHOT ' to identify in development
- GAV Syntax: groupId:artifactId:version

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.lds.training</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
</project>
```

Packaging

- Build type identified using the "packaging" element
- Tells Maven how to build the project
- Example packaging types:
  - pom, jar, war, ear, custom
  - Default is jar

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <groupId>org.lds.training</groupId>
    <version>1.0</version>
    <packaging>jar</packaging>
</project>
```

Project Inheritance

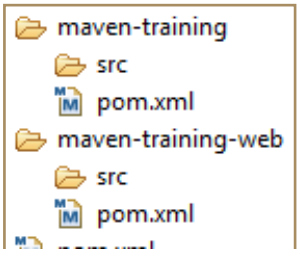
- Pom files can inherit configuration
  - groupId, version
  - Project Config
  - Dependencies
  - Plugin configuration
  - Etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <parent>
        <artifactId>maven-training-parent</artifactId>
        <groupId>org.lds.training</groupId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <packaging>jar</packaging>
</project>
```

Multi Module Projects

- Maven has 1st class multi-module support
- Each maven project creates 1 primary artifact
- A parent pom is used to group modules

```
<project>
    ...
    <packaging>pom</packaging>
    <modules>
        <module>maven-training</module>
        <module>maven-training-web</module>
    </modules>
</project>
```



Maven Conventions

- Maven is opinionated about project structure
  - target: Default work directory
  - src: All project source files go in this directory
  - src/main: All sources that go into primary artifact
  - src/test: All sources contributing to testing project
  - src/main/java: All java source files
  - src/main/webapp: All web source files
  - src/main/resources: All non compiled source files
  - src/test/java: All java test source files
  - src/test/resources: All non compiled test source files

Maven Build Lifecycle

- A Maven build follow a lifecycle
- Default lifecycle
  - generate-sources/generate-resources
  - compile
  - test
  - package
  - integration-test (pre and post)
  - Install
  - deploy
- There is also a Clean lifecycle

Example Maven Goals

- To invoke a Maven build you set a lifecycle “goal”
- mvn install
  - Invokes generate* and compile, test, package, integration-test, install
- mvn clean
  - Invokes just clean
- mvn clean compile
  - Clean old builds and execute generate*, compile
- mvn compile install
  - Invokes generate*, compile, test, integration-test, package, install
- mvn test clean
  - Invokes generate*, compile, test then cleans

Maven and Dependencies

- Maven revolutionized Java dependency management
  - No more checking libraries into version control
- Introduced the Maven Repository concept
  - Established Maven Central
- Created a module metadata file (POM)
- Introduced concept of transitive dependency
- Often include source and javadoc artifacts

Adding a Dependency

- Dependencies consist of:
  - GAV
  - Scope: compile, test, provided (default=compile)
  - Type: jar, pom, war, ear, zip (default=jar)

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

Maven Repositories

- Dependencies are downloaded from repositories
  - Via http
- Downloaded dependencies are cached in a local repository
  - Usually found in ${user.home}/.m2/repository
- Repository follows a simple directory structure
  - {groupId}/{artifactId}/{version}/{artifactId}-{version}.jar
  - groupId '.' is replaced with '/'
- Maven Central is primary community repo
  - http://repo1.maven.org/maven2

Maven Demo

MSBuild

- Microsoft Build Engine, better known as MSBuild, is a free and open-source build tool set for managed code as well as native C++ code and was part of .NET Framework. Visual Studio depends on MSBuild, but not the vice versa.

```
<Project DefaultTargets="Build" ToolsVersion="16.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup>
    <ProjectConfiguration Include="Debug|Win32">
      <Configuration>Debug</Configuration>
      <Platform>Win32</Platform>
    </ProjectConfiguration>
    <ProjectConfiguration Include="Release|Win32">
      <Configuration>Release</Configuration>
      <Platform>Win32</Platform>
    </ProjectConfiguration>
  </ItemGroup>
  <Import
Project="$(VCTargetsPath)\Microsoft.Cpp.default.props" />
  <PropertyGroup>
    <ConfigurationType>Application</ConfigurationType>
    <PlatformToolset>v142</PlatformToolset>
  </PropertyGroup>
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props"
/>
  <ItemGroup>
    <ClCompile Include="main.cpp" />
  </ItemGroup>
  <ItemGroup>
    <ClInclude Include="main.h" />
  </ItemGroup>
  <Import
Project="$(VCTargetsPath)\Microsoft.Cpp.Targets" />
</Project>
```

SonarQube

- SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells on 20+ programming languages

SonarQube Demo

With Maven

Web Server

Web Server

- A web server is a computer/software that runs websites. It's a computer program that distributes web pages as they are requisitioned. The basic objective of the web server is to store, process and deliver web pages to the users. This intercommunication is done using Hypertext Transfer Protocol (HTTP).

Popular Web Servers

- Apache Web Server
- Microsoft's Internet Information Services (IIS)
- Nginx

Apache Tomcat

- Web Server + Web Container
- Apache Tomcat an open-source Java servlet container that implements many Java Enterprise Specs such as the Websites API, Servlets and Java-Server Pages
- Began as the reference implementation for servlet/jsp in 1998
- It is still one of the most widely used java-sever due to several capabilities such as good extensibility, proven core engine, and well-tested and durable.

Demo

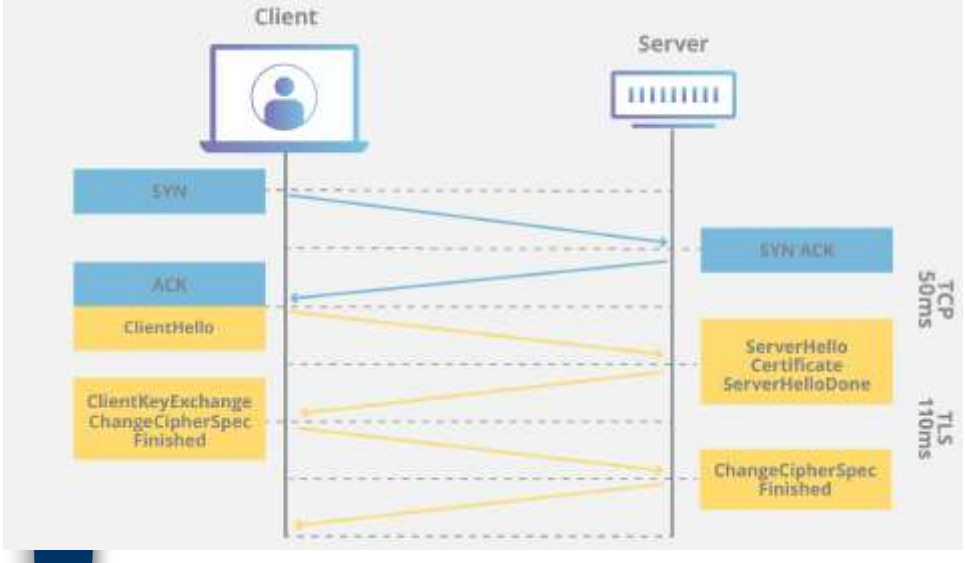
- Start/Stop tomcat server
- Deploy war file
  - Using eclipse
  - Without eclipse

Security

- Web security is important to keeping hackers and cyber-thieves from accessing sensitive information. Without a proactive security strategy, businesses risk the spread and escalation of malware, attacks on other websites, networks, and other IT infrastructures.
- Network Security

Demo

- Generate SSL Certificate
- Tomcat configuration of SSL Certificate



Git

Git

- As **Git** is a distributed version control system, it can be used as a server out of the box. Dedicated **Git** server software helps, amongst other features, to add access control, display the contents of a **Git** repository via the web, and help managing multiple repositories.

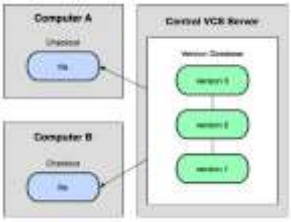
Version Control Systems

- Version Control (or Revision Control, or Source Control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all “real” projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why Version Control?

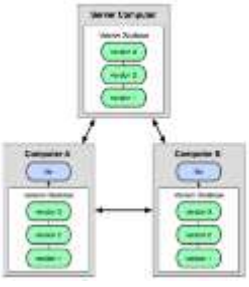
- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes

Centralized VCS



- In Subversion, CVS, Perforce, etc.
  - A central server repository (repo) holds the "official copy" of the code
  - The server maintains the sole version history of the repo
- You make "checkouts" of it to your local copy
  - You make local modifications
  - Your changes are not versioned
- When you're done, you "check in" back to the server
  - your checkin increments the repo's version

Distributed VCS (Git)



- In git, mercurial, etc., you don't "checkout" from a central repo
  - You "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - Yours is "just as good" as theirs
- Many operations are local:
  - Check in/out from local repo
  - Commit changes to local repo
  - Local repo keeps version history
- When you're ready, you can "push" changes back to server

Why Git?

- Git has many advantages over earlier systems
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree
  - Very Popular

Version Control Terminology

- Version Control System (VCS) or (SCM)
- Repository
- Commit
- SHA
- Working Directory
- Checkout
- Staging Area/Index
- Branch

Version Control Terminology

- Version Control System :
  - A VCS allows you to: revert files back to a previous state, revert the entire project back to a previous state, review changes made over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.
- Repository:
  - A directory that contains your project work which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer.

Version Control Terminology

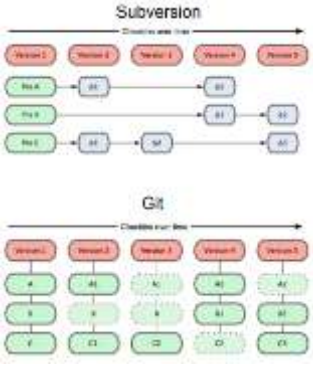
- Commit
  - Git thinks of its data like a set of snapshots of a mini file system.
  - Think of it as a save point during a video game.
- SHA
  - A SHA is basically an ID number for each commit.
  - Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6
- Branch
  - A branch is when a new line of development is created that diverges from the main line of development. This alternative line of development can continue without altering the main line.

Version Control Terminology

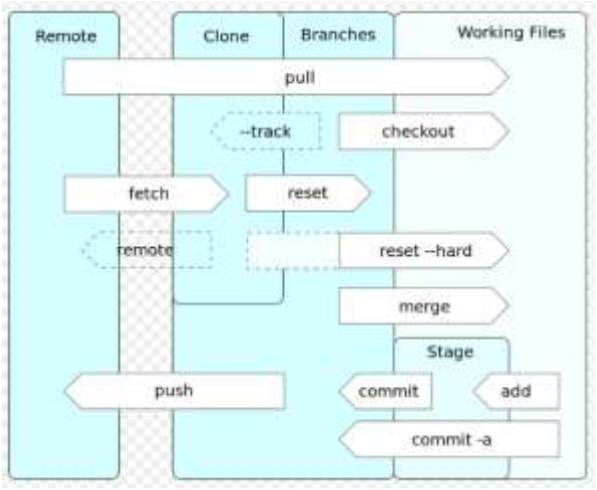
- Working Directory
  - files that you see in your computer's file system. When you open project files up on a code editor, you're working with files in the Working Directory.
- Checkout
  - Content in the repository has been copied to the Working Directory. Possible to checkout many things from a repository; a file, a commit, a branch, etc.
- Staging Area
  - You can think of the staging area as a prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repo

Git

- Centralized VCS like Subversion track version data on each individual file.
- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.



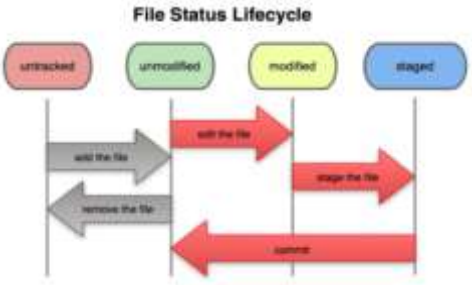
Git



In your local copy on git, files can be:

- In your local repo
  - (committed)
- Checked out and modified, but not yet committed
  - (working copy)
- Or, in-between, in a "staging" area
  - Staged files are ready to be committed.
  - A commit saves a snapshot of all staged state.

Basic Git Workflow



- Modify files in your working directory.
- Stage files, adding snapshots of them to your staging area.
- Commit, which takes the files in the staging area and stores that snapshot permanently to your Git directory.

Initial Git configuration

- Set the name and email for Git to use when you commit:
  - git config --global user.name “.."
  - git config --global user.email e@gmail.com
  - You can call git config –list to verify these are set.
- Set the editor that is used for writing commit messages:
  - git config --global core.editor nano
    - (it is vim by default)

Creating a Git Repo

- To create a new local Git repo in your current directory:
  - git init
    - This will create a .git directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - git add filename
  - git commit –m "commit message"
- To clone a remote repo to your current directory:
  - git clone url localDirectoryName
    - This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your local repo)

Git Commands

command	description
`git clone url [dir]`	copy a Git repository so you can add to it
`git add file`	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help [command]`	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

Add and commit a file

- The first time we ask a file to be tracked, and every time before we commit a file, we must add it to the staging area:
  - git add Hello.java Goodbye.java
    - Takes a snapshot of these files, adds them to the staging area.
- To move staged changes into the repo, we commit:
  - git commit –m "Fixing bug #22"
- To undo changes on a file before you have committed it:
  - git reset HEAD -- filename (unstages the file)
  - git checkout -- filename (undoes your changes)
  - All these commands are acting on your local version of repo.

Viewing/undoing changes

- To view status of files in working directory and staging area:
  - git status or git status –s (short version)
- To see what is modified but unstaged:
  - git diff
- To see a list of staged changes:
  - git diff --cached
- To see a log of all changes in your local repo:
  - git log or git log --oneline (shorter version)
  - git log -5 (to show only the 5 most recent updates) etc

Branching and Merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:
  - git branch name
- To list all local branches: (* = current branch)
  - git branch
- To switch to a given local branch:
  - git checkout branchname
- To merge changes from a branch into the local master:
  - git checkout master
  - git merge branchname

Merge Conflicts

The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

branch 1's version

branch 2's version

Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

Interaction with Remote Repo

- Push your local changes to the remote repo.
- Pull from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)
- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - git pull origin master
- To put your changes from your local repo in the remote repo:
  - git push origin master

GitHub

- GitHub.com is a site for online storage of Git repositories.
  - You can create a remote repo there and push code to it.
  - Many open source projects use it, such as the Linux kernel.
  - You can get free space for open source projects, or you can pay for private projects.

Demo

- Create git repository
- Check status
- Commit Changes
- Observe Issues

Jenkins

Continuous Integration (CI)

- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.
- By integrating regularly, you can detect errors quickly, and locate them more easily.

Continuous Integration

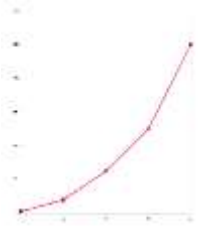
- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

-- Martin Fowler

Ref: http://martinfowler.com/articles/continuousIntegration.html

Why Continuous Integration?

- Integration is hard, effort increase exponentially with
  - Number of components
  - Number of bugs
  - Time since last integration



Ref: http://www.slideshare.net/carlo.bonamico/continuous-integration-with-hudson

CI – What does it really mean?

- At a regular frequency (ideally at every commit), the system is:
  - Integrated
    - All changes up until that point are combined into the project
  - Built
    - The code is compiled into an executable or package
  - Tested
    - Automated test suites are run
  - Archived
    - Versioned and stored, can be distributed as is, if desired
  - Deployed
    - Loaded onto a system where the developers can interact with it

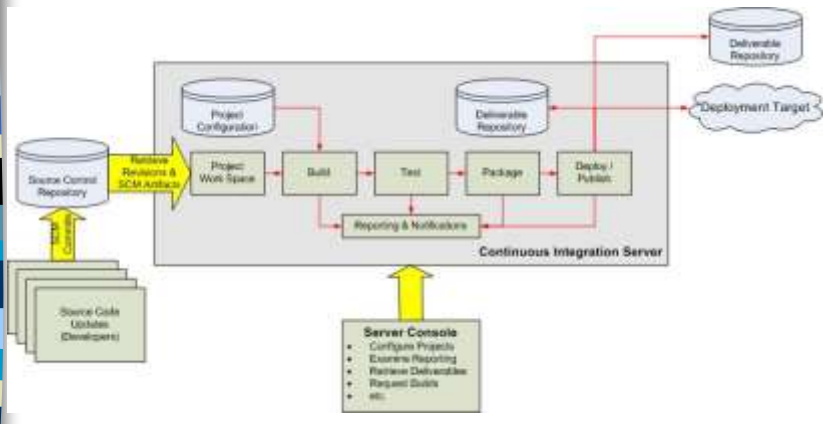
Continuous Integration Benefit

- Project Management
  - Detect system development problems earlier
  - Reduce risks of cost, schedule, and budget
- Code Quality
  - Measurable and visible code quality
  - Continuous automatic regression unit test

Best Practices

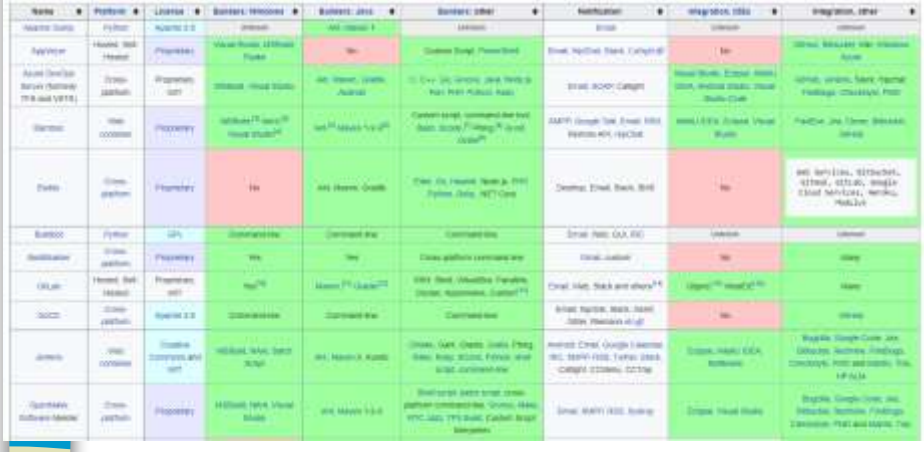
- Single Source Repository
- Automate the Build and Test
- Everyone Commits Every Day
- Keep the Build Fast
- Everyone can see what's happening
- Automate Deployment (Optional)

Continuous Integration Overview



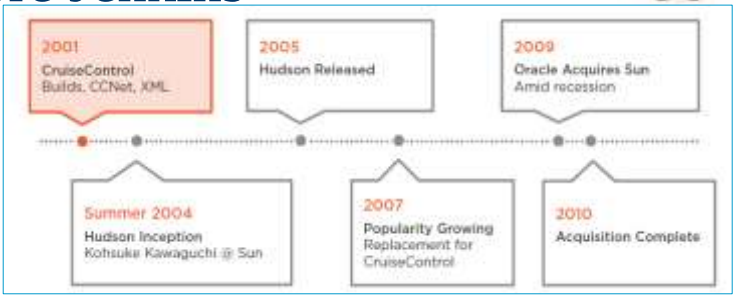
Ref: http://www.javaworld.com/javaworld/jw-12-2008/images/CIOverview.jpg

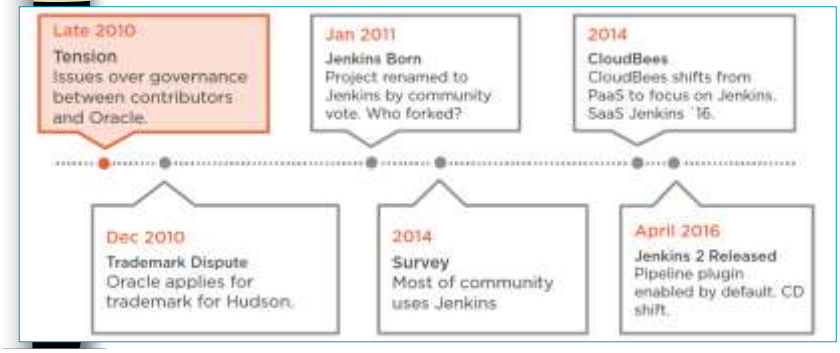
Continuous Integration Tools



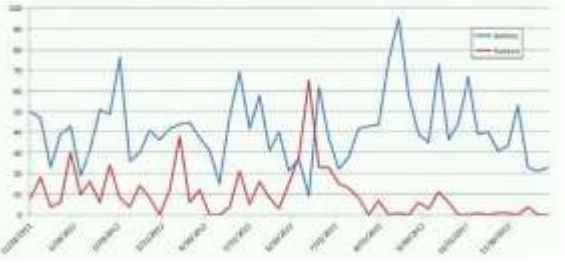
Ref: http://en.wikipedia.org/wiki/Comparison_of_Continuous_Integration_Software

Before Jenkins

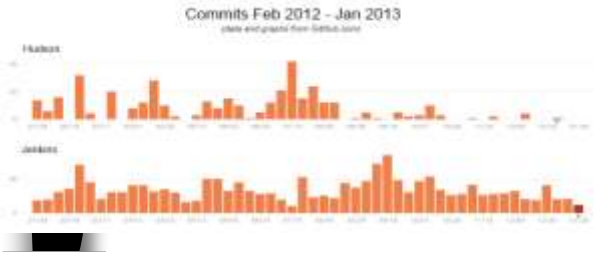




What Happened Next?



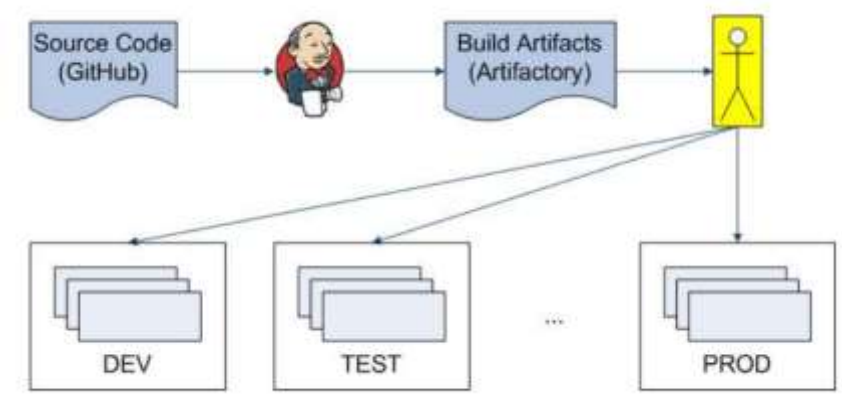
2012, THE DECISIVE YEAR



Standard Software Platform

- Started platform definition in 2011
  - Homogeneous by default
- Tools
  - Java, Spring, Tomcat, Postgres
  - Git/GitHub, Gradle, Jenkins, Artifactory, Liquibase
- Process
  - Standard development workflow
  - Standard application shape & operational profile

Initial Delivery Pipeline



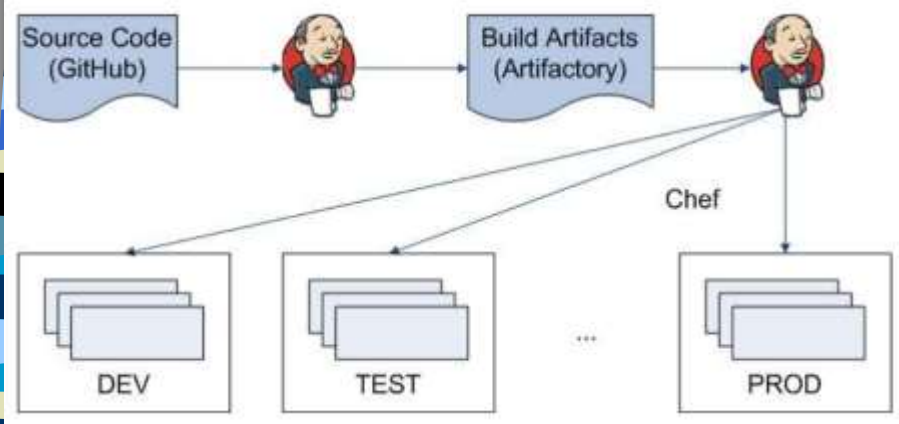
Initial Delivery Pipeline

- Automated build process
- Publish build artifacts to Artifactory
  - Application WARs
  - Liquibase JARs
- Manual deploys
  - (Many apps) x (many versions) x (multiple environments) = TIME & EFFORT
  - The more frequently a task is performed, the greater the return from improved efficiency

Improved Deployment Process

- Goals
  - Reduce effort
  - Improve speed, reliability, and frequency
  - Handle app deploys and db schema updates
  - Enable self-service
- Process Changes
  - Manual -> Automated
  - Prose instructions -> Infrastructure as code

Target Delivery Pipeline



Jenkins Features

- Trigger a build
- Get source code from repository
- Automatically build and test
- Generate report & notify
- Deploy
- Distributed build

Jenkins Requirement

- Web Server (Tomcat, WebLogic, …)
- Build tool (Maven, Ant)
- SCM (Git, Svn, Cvs, …)

Jenkins Plugins

- Build triggers
- Source code management
- Build tools
- Build wrappers
- Build notifiers
- Build reports
- Artifact uploaders
- UI plugins
- Authentication and user management

Build Trigger

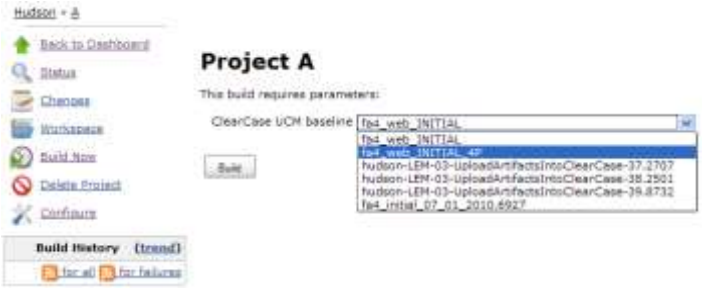
- Manually click build button
- Build periodically
- Build whenever a SNAPSHOT dependency is built
- Build after other projects are built
- Poll SCM
- IRC, Jabber, …

Get Source Code (1/2)

- CVS (build-in)
- SVN (build-in)
- GIT (requires Git)
- ClearCase (requires ClearCase)
- Mercurial, PVCS, VSS, …

Get Source Code (2/2)

- Get current snapshot
- Get baseline (tag)



Code Change History

Build Tools

- Java
  - Maven (build-in), Ant, Gradle
- .Net
  - MSBuild, PowerShell
- Shell script
  - Python, Ruby, Groovy

Build Wrapper

- Build name (version no) setter
- Virtual machine (VMWare, Virtual Box)
- Set environment variable
- ClearCase release plugin
- …

Build Notifier

- E-mail
- Twitter
- Jabber
- IRC
- RSS
- Google calendar
- …

Build Report

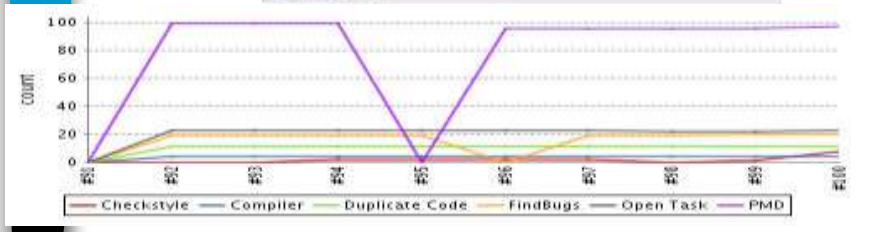
- Static Code Analysis
  - Checkstyle, PMD, Findbugs, Compiler Warning
- Test Report & Code Coverage
  - JUnit, TestNG, Cobertura, Clover
- Open Tasks

Static Code Analysis

All DashBoard +

Warnings per Job

Job	Checkstyle	Duplicate Code	FindBugs	PMD	Open Tasks	Compiler Warnings	Total
M-Multi-Freestyle	14	-	0	86	24	0	124
M-Multi-Maven	14	11	12	80	0	4	121
M-Single-Freestyle	0	0	15	18	99	3	135
M-Single-Freestyle-Site	5967	2	-	18	5	0	5992
M-Single-Maven	-	-	-	-	121	0	121
M-Single-Maven-2	-	4	18	243	0	0	267
M-Single-Maven-Site	1784	2	73	2	0	-	1860



CheckStyle

CheckStyle Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
6766	2	0

Summary

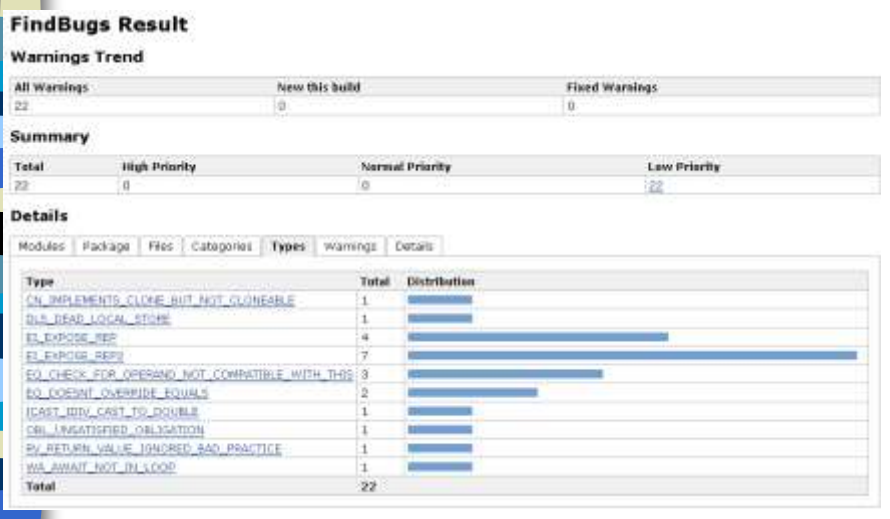
Total	High Priority	Normal Priority	Low Priority
6766	6766	0	0

Details

Modules | Package | Files | Categories | Types | Warnings | Details | New

Type	Total	Distribution
AvoidInlineConditionalsCheck	47	
AvoidNestedBlocksCheck	2	
AvoidStarImportCheck	1	
ConstantNameCheck	11	
DesignForExtensionCheck	799	
EmptyBlockCheck	29	
FinalClassCheck	9	
FinalParametersCheck	980	
HiddenFieldCheck	399	
HideUtilityClassConstructorCheck	9	
InnerAssignmentCheck	1	
InterfaceIsTypeCheck	8	
JavadocMethodCheck	1472	

FindBugs



FindBugs Result

Warnings Trend

All Warnings	New this build	Fixed Warnings
22	0	0

Summary

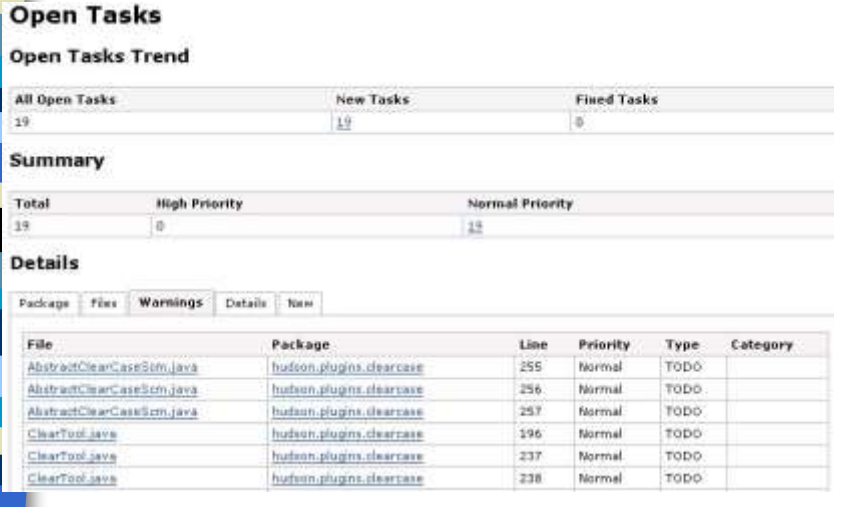
Total	High Priority	Normal Priority	Low Priority
22	0	0	22

Details

Modules | Package | Files | Categories | Types | Warnings | Details

Type	Total	Distribution
CN_IMPLEMENTS_CLONE_BUT_NOT_CLONEABLE	1	
DLS_DEAD_LOCAL_STORE	1	
EI_EXPOSE_REP	4	
EI_EXPOSE_REP2	7	
EQ_CHECK_FOR_OPERAND_NOT_COMPATIBLE_WITH_THIS	3	
EQ_DOESNT_OVERRIDE_EQUALS	2	
ICAST_IDIV_CAST_TO_DOUBLE	1	
OBL_UNSATISFIED_OBLIGATION	1	
RV_RETURN_VALUE_IGNORED_BAD_PRACTICE	1	
WA_AWAIT_NOT_IN_LOOP	1	
Total	22	

Open Tag



Open Tasks

Open Tasks Trend

All Open Tasks	New Tasks	Fixed Tasks
19	19	0

Summary

Total	High Priority	Normal Priority
19	0	19

Details

Package | Files | Warnings | Details | New

File	Package	Line	Priority	Type	Category
AbstractClearCaseScm.java	hudson.plugins.clearcase	255	Normal	TODO	
AbstractClearCaseScm.java	hudson.plugins.clearcase	256	Normal	TODO	
AbstractClearCaseScm.java	hudson.plugins.clearcase	257	Normal	TODO	
ClearTool.java	hudson.plugins.clearcase	196	Normal	TODO	
ClearTool.java	hudson.plugins.clearcase	237	Normal	TODO	
ClearTool.java	hudson.plugins.clearcase	238	Normal	TODO	

Duplicate Code

Duplicate Code Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
10	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
10	0	10	0

Details

Package | Files | Warnings | Details | Normal | Low

File	Number of lines	Duplicated in
NetscapeDraftSpec.java:120	17	BestMatchSpec.java:117
BestMatchSpec.java:117	17	NetscapeDraftSpec.java:120
SSLSocketFactory.java:462	28	PlainSocketFactory.java:154
AbstractAuthenticationHandler.java:82	23	AuthSchemeBase.java:88
RequestProxyAuthentication.java:95	28	RequestTargetAuthentication.java:86
RFC2109DomainHandler.java:47	32	BasicDomainHandler.java:45
AuthSchemeBase.java:88	23	AbstractAuthenticationHandler.java:82
NetscapeDraftSpec.java:127	19	BrowserCompatSpec.java:163
PlainSocketFactory.java:154	28	SSLSocketFactory.java:462
RequestTargetAuthentication.java:86	28	RequestProxyAuthentication.java:95
BrowserCompatSpec.java:163	19	NetscapeDraftSpec.java:127
DefaultRedirectHandler.java:141	44	DefaultRedirectStrategy.java:133
BrowserCompatSpec.java:129	34	BestMatchSpec.java:101
BasicDomainHandler.java:45	32	RFC2109DomainHandler.java:47
DefaultRedirectStrategy.java:133	44	DefaultRedirectHandler.java:141
BestMatchSpec.java:101	34	BrowserCompatSpec.java:129

Test Report

Tests

Job	Success		Failed		Skipped		Total
	#	%	#	%	#	%	#
Eden-AdvecWords	127	100%	0	0%	0	0%	127
Eden-Eden	266	100%	0	0%	0	0%	266
Eden-Monditi-amicimobservice	155	100%	0	0%	0	0%	155
Eden-Pabri	20	100%	0	0%	0	0%	20
Total	568	100%	0	0%	0	0%	568

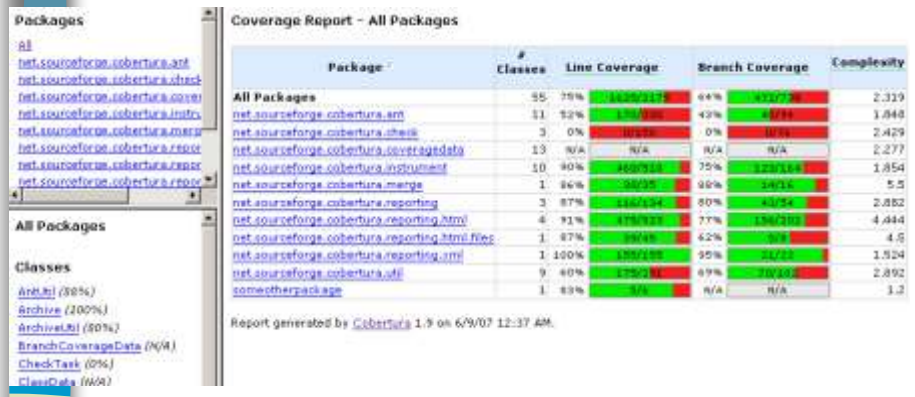
Test Result

0 failures (±0) , 2 skipped (±0)

1,513 tests (±0)

Module	Fail	(diff)	Total	(diff)
org.apache.httpcomponents:httpclient	0		582	
org.apache.httpcomponents:httpclient-cache	0		920	
org.apache.httpcomponents:httpmime	0		11	

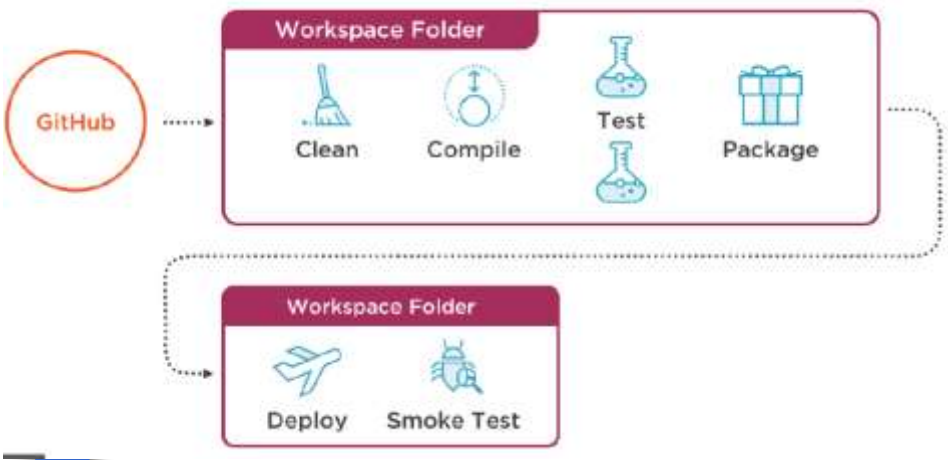
Test Code Coverage



Ref: http://cobertura.sourceforge.net/sample/

What is Jenkins Pipeline

Delivery Pipeline



Jenkins Pipeline

- Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
- A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment

Jenkinsfile

- The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository.
- This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.
- Pipeline domain-specific language (DSL) syntax

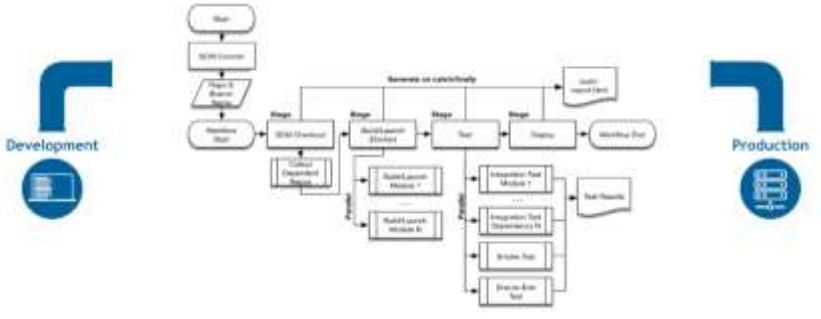
Declarative Vs Scripted Pipeline syntax

- Declarative and Scripted Pipelines are constructed fundamentally differently.
- Declarative Pipeline is a more recent feature of Jenkins Pipeline which:
  - Provides richer syntactical features over Scripted Pipeline syntax
  - Designed to make writing and reading Pipeline code easier.
- Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline.

Why Pipeline?

- Code
  - Pipelines are implemented in code and typically checked into source control
- Durable
  - can survive planned and unplanned restarts of the Jenkins master.
- Pausable
  - Pipelines can stop and wait for human input or approval before continuing the Pipeline run.
- Versatile
  - Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- Extensible
  - The Pipeline plugin supports custom extensions to its DSL and multiple options for integration with other plugins.

Development → Production



Pipeline Concepts

- Pipeline
  - A Pipeline is a user-defined model of a CD pipeline.
  - Defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.
  - A pipeline block is a key part of Declarative Pipeline syntax.
- Node
  - A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.
  - Also, a node block is a key part of Scripted Pipeline syntax.
- Stage
  - A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress.
- Step
  - A single task. Fundamentally, a step tells Jenkins what to do at a particular point in time (or "step" in the process).

Demo

- A project deployment on Jenkins

GitLab

Why GitLab?

- GitHub is a collaboration platform that helps review and manage codes remotely, **GitLab is majorly focused on DevOps and CI/CD**. GitHub is more popular amongst the developers as it holds millions of repositories, but recently GitLab has been gaining popularity, as the company continues to add new features to make it more competitive and user-friendly.

GitLab IDE

- Most work in GitLab is done in a project, where files and source code are stored. The Web IDE is a feature that allows you to edit the code directly in your browser rather than locally. The Web IDE can be opened with button on using the period key shortcut form anywhere on the repository page.

GitLab Vocabulary

- Projects
  - Create projects to host your codebase
  - Use projects to track issues, plan work, collaborate on code, and continuously build, test, and use built-in CI/CD to deploy your app
  - Can be made available publicly, internally, or privately

GitLab Vocabulary

- Groups
  - GitLab groups allow you to group projects into directories and give users access to several projects at once
- Repository
  - Repository is where you store your code and make changes to it. Your changes are tracked with version control.
  - Each project contains a repository.

GitLab Vocabulary

- Branch
  - A branch is a version of a project's working tree.
  - Create a branch for each set of related changes you make. This keeps each set of changes separate from each other, allowing changes to be made in parallel, without affecting each other.
  - Give read/write permissions per branch

GitLab Vocabulary

- Protected branches
  - To impose further restrictions on certain branches, they can be protected(in addition to basic read/write)
  - The default branch for your repository is protected by default.
  - Users in maintainer role can modify a protected branch

GitLab IDE Demo

- Create a project
- Modify Repository
  - Add / remove files
  - Understand Staging and Committing
  - Change protection
  - Observe difference
  - Watch earlier commits, history
  - Understand forking and merging

GitLab CI/CD Concepts

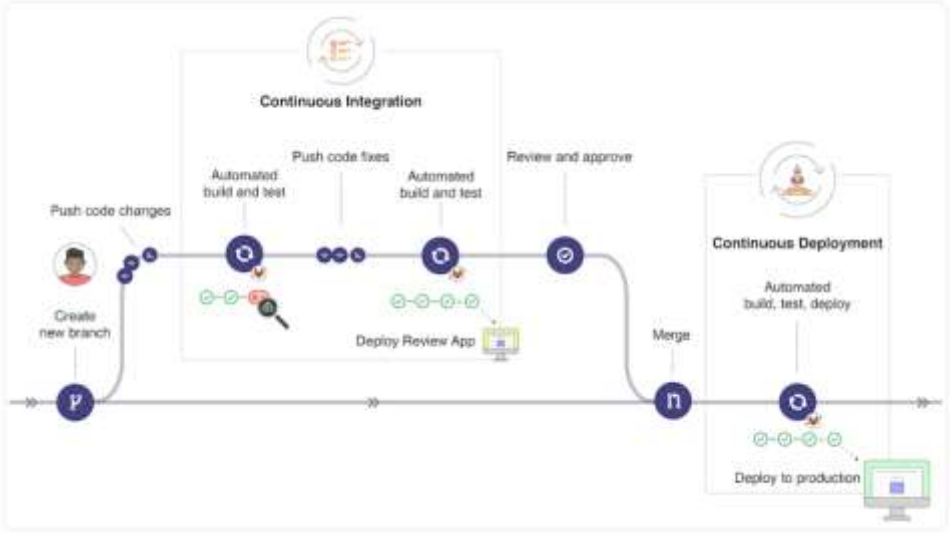
- Continuous Integration
  - For application with Git Repository code in GitLab, developers push code changes every hour/day.
  - For every push to the repository, create a set of scripts to build and test your application automatically.
  - These scripts help decrease the chances that you introduce errors in your application.

Each change submitted to an application, even to development branches, is built and tested automatically and continuously.

GitLab CI/CD Concepts

- Continuous Delivery
  - Is a step beyond Continuous Integration.
  - In addition to build and test each time a code change is pushed to the codebase, the application is also deployed continuously.
  - However, with continuous delivery, you trigger the deployments manually.
- Continuous Deployment
  - Similar to Continuous Delivery. The difference is that instead of deploying your application manually, you set it to be deployed automatically. Human intervention is not required.

Continuous Delivery checks the code automatically, but it requires human intervention to manually and strategically trigger the deployment of the changes.



GitLab Pipelines

- Two main components
  - Jobs - Describe the tasks that need to be done (compile, test, deploy etc)
  - Stages - Define the order in which jobs will be executed
- Set of Instructions for a program to execute
- GitLab Runner - Program to executes jobs Gitlab pipeline

GitLab Runner

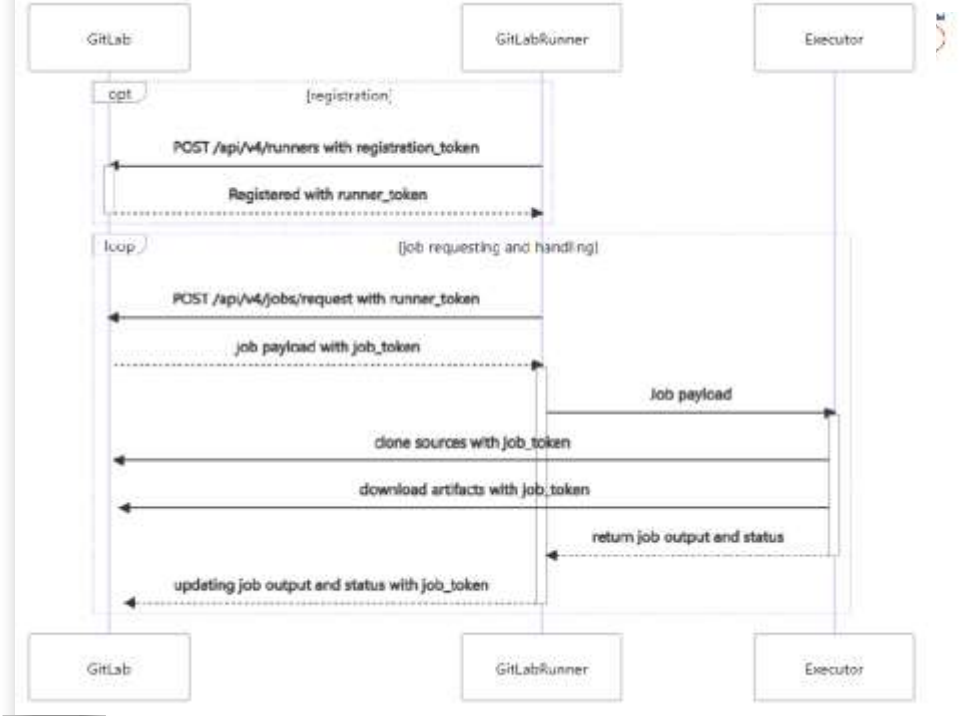
- Separate program that can be run on your local host, vm or even container
- Similar to jenkins Agent
- GitLab assigns pipeline jobs to available runners at runtime

GitLab Runner

- GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline.
- Install the GitLab Runner application on infrastructure that you own or manage.
- GitLab Runner is open-source and written in Go. It can be run as a single binary; no language-specific requirements are needed.

GitLab Runner Features

- Run multiple jobs concurrently.
- Use multiple tokens with multiple servers (even per-project).
- Limit the number of concurrent jobs per-token.
- Jobs can be run:
  - Locally.
  - Using Docker containers.
  - Using Docker containers and executing job over SSH.
  - Using Docker containers with autoscaling on different clouds and virtualization hypervisors.
- Connecting to a remote SSH server.



Demo

- Create pipeline using editor
- Pipeline
  - Jobs
  - Stages
  - Scripts
  - Artifacts
  - Variables
    - Masked
    - Protected

Demo

- Download GitLab Runner
- See configuration files for the same
  - Global Section
  - Local
  - https://docs.gitlab.com/runner/configuration/advanced-configuration.html (options for logging/ concurrency etc)

GitLab for Collaboration

Collaboration Tools

An example flow using a disparate set of collaboration tools.

With GitLab

An example workflow using Figma (or Sketch) and GitLab.

Why GitLab?

- A single source of truth For the entire team
- Teams are much more efficient when everyone can work within the same tool at the same time throughout the entire process to produce a single source of truth.
- No more lost feedback or decision juggling acts. Everyone is on the same page and teams can focus less on process and more on building great products.

Plan and Track Work

- Issues
  - Use issues to collaborate on ideas, solve problems, and plan work. Share and discuss proposals with your team and with outside collaborators.
  - You can use issues for many purposes, customized to your needs and workflow.
    - Discuss the implementation of an idea.
    - Track tasks and work status.
    - Accept feature proposals, questions, support requests, or bug reports.
    - Elaborate on code implementations.

Plan and Track Work

- Milestones
  - Milestones in GitLab are a way to track issues and merge requests created to achieve a broader goal in a certain period of time.
  - Milestones allow you to organize issues and merge requests into a cohesive group, with an optional start date and an optional due date.
  - Milestones as Agile sprints
  - Milestones as releases

Plan and Track Work

- To-Do List
  - Your To-Do List is a chronological list of items waiting for your input. The items are known as to-do items.
  - You can use the To-Do List to track actions related to:
    - Issues
    - Merge requests
    - Epics
    - Designs

Continuous Delivery/Deployment

GitLab Continuous Delivery

- Performs all the steps to deploy your code to your production environment including provisioning infrastructure, managing changes via version control, ticketing and release versioning, progressively deploying code, verifying and monitoring those changes and providing the ability to roll back when necessary - all from the same application that also hosts your source code and helps with Continuous Integration.

Introducing environments

- All you need to do it to specify the corresponding environment for each deployment job

```
variables:
  S3_BUCKET_NAME: "yourbucket"

deploy to production:
  environment: production
  image: python:latest
  script:
  - pip install awscli
  - aws s3 cp ./ s3://$S3_BUCKET_NAME/ -
-recursive --exclude "*" --include "*.html"
  only:
  - master
.....
```

Auto DevOps

- GitLab Auto DevOps is a collection of pre-configured features and integrations that work together to support your software delivery process.
- Auto DevOps features and integrations:
  - Detect your code's language.
  - Build and test your application.
  - Measure code quality.
  - Scan for vulnerabilities and security flaws.
  - Check for licensing issues.
  - Monitor in real time.
  - Deploy your application.

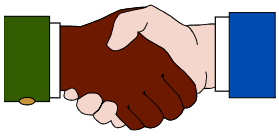
Monitoring

- Difference between monolith and micro services monitoring
  - Logging
  - Tracing
  - Metrices
  - Trouble Shooting

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com