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

A Fast AND Steady Approach

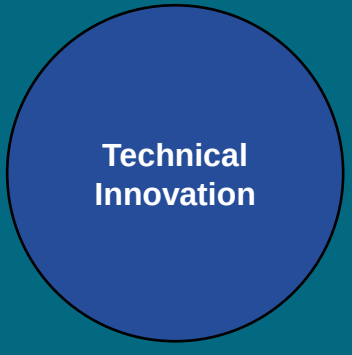
Discuss

Why Change?

fands™
A Fast AND Steady Approach

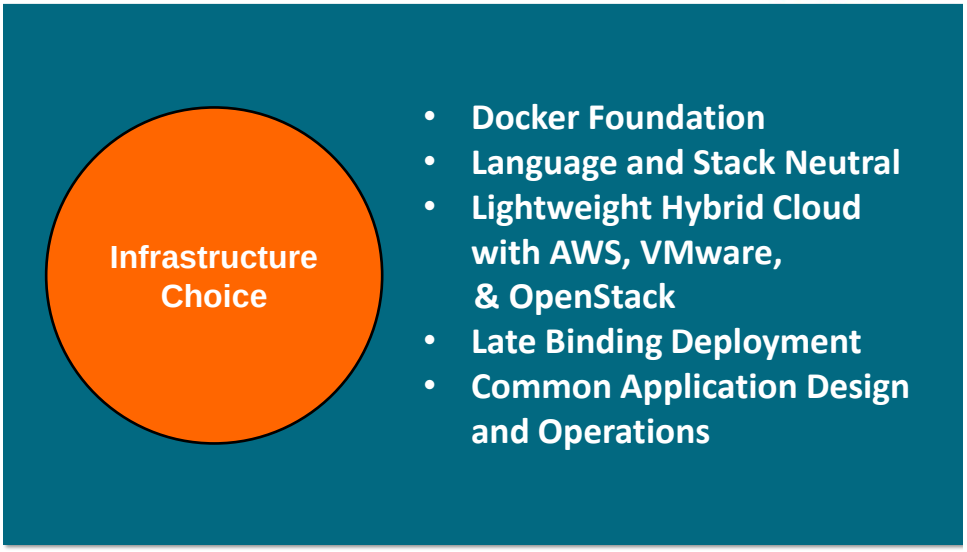
- Time-to-Market Acceleration
- Experimentation
- Rapid Prototyping
- Flexible Partnering
- IoT/IoS Support

fands™
A Fast AND Steady Approach

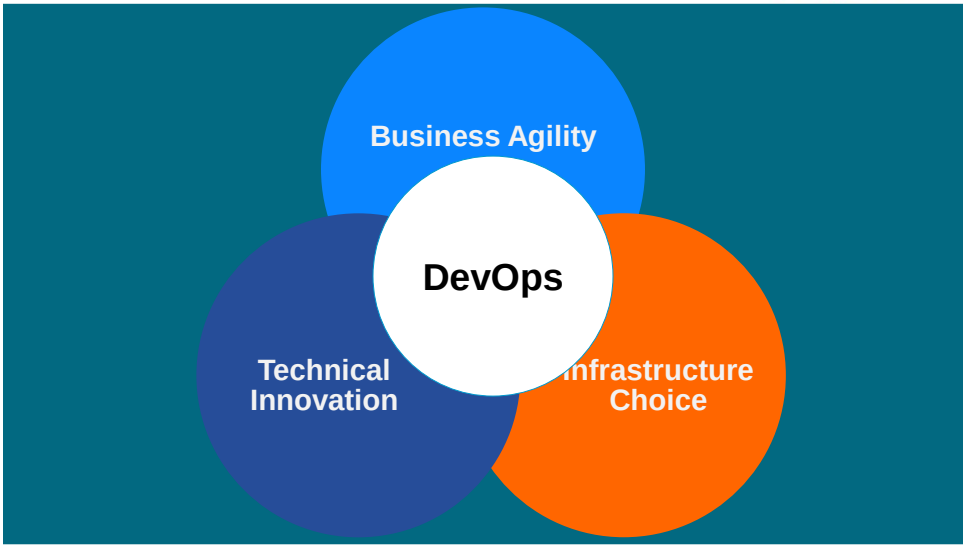


- Polyglot Enablement
- DevOps Automation
- API Support
- Microservices Architecture
- Blue-Green Deployment
- Application Scaling and Elasticity
- PaaS

fands™
A Fast AND Steady Approach



fands™
A Fast AND Steady Approach



Monoliths to Micro Services

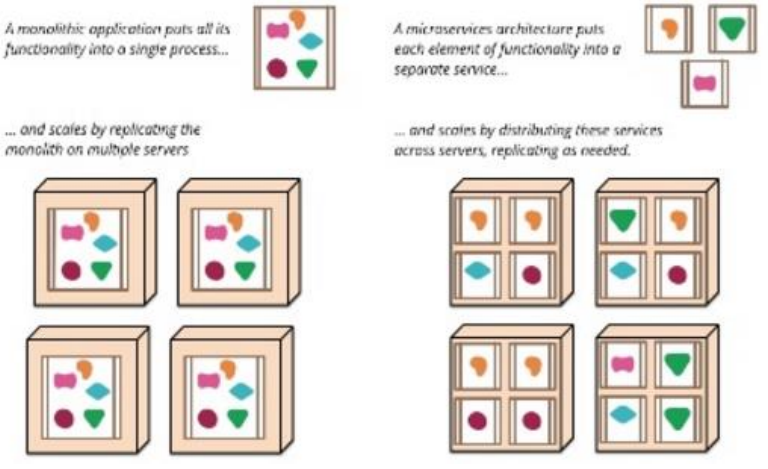
Monoliths to Micro Services

- 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



fands™

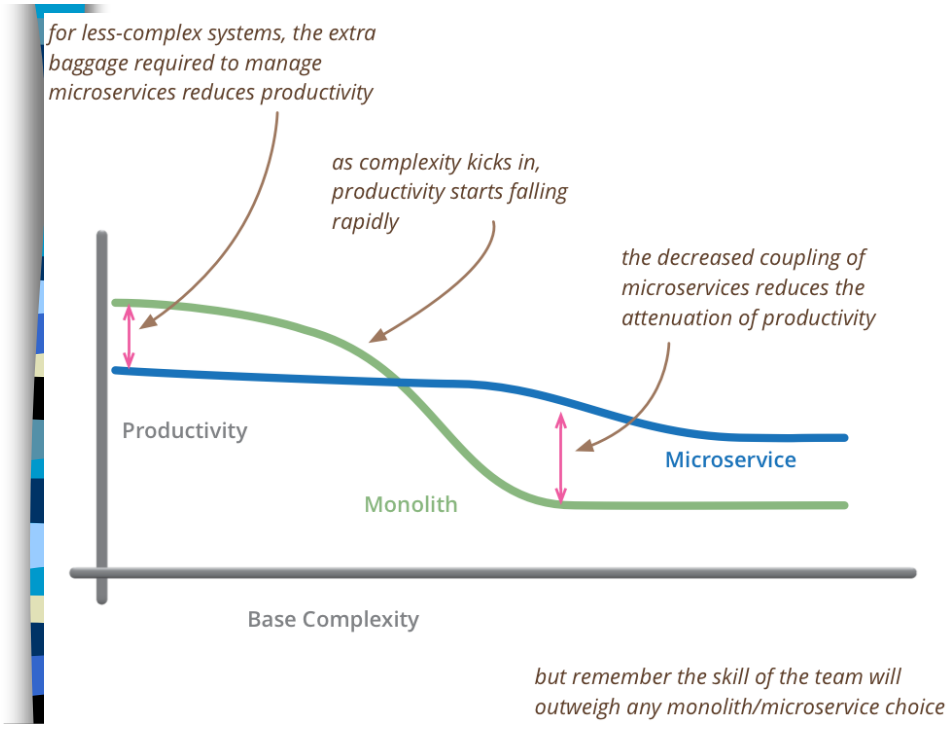
Monoliths Vs Micro Services

- 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

fands™

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

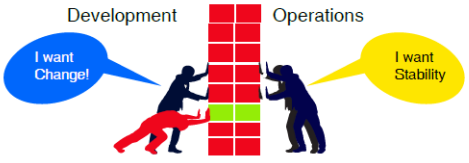


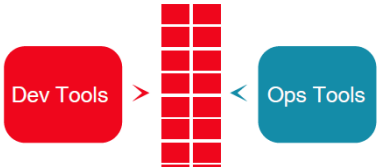
fands™

Issues with Micro Services

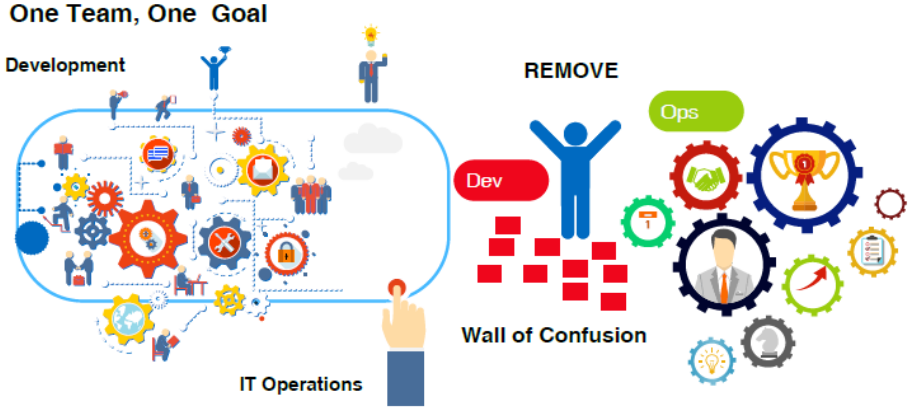
- Network overhead
- Transaction coordination
- Need for duplicating common data
  - keeping it in sync
- Complicated deployment pipeline dependencies

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.

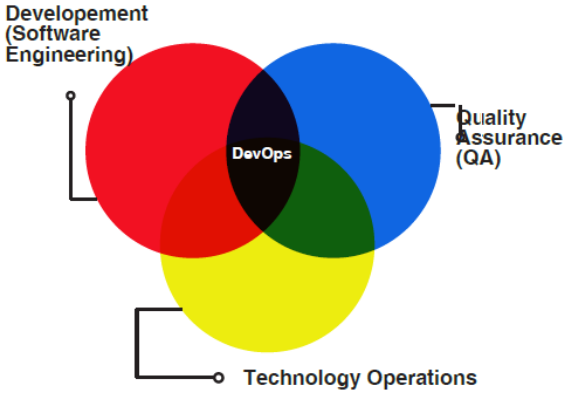
DevOps

DevOps is a way of thinking.

Culture	• Hearts & Minds • Embrace Change
Automation	• CI/CD • "Infrastucture as Code"
Lean	• Focus on producing for the end-user • Small batch sizes
Metrics	• Measure everything • Show the Improvement
Sharing	• Open information sharing • Collaboration

CALMS Model

DevOps



fands™

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

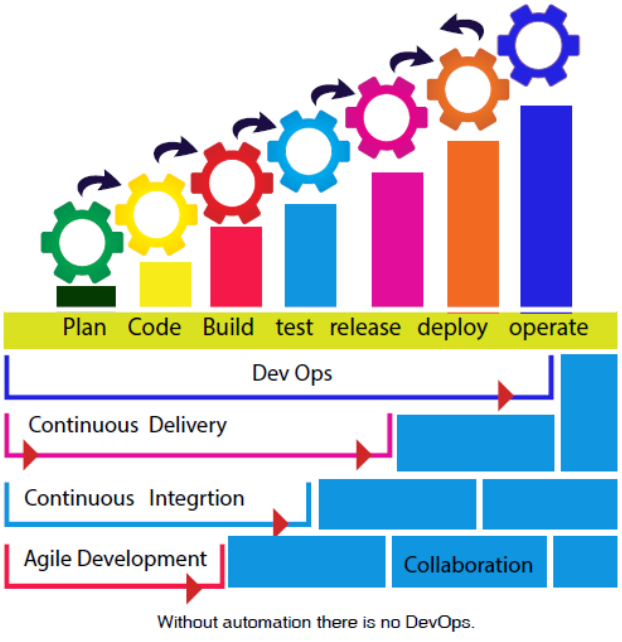
fands™

DevOps Combines

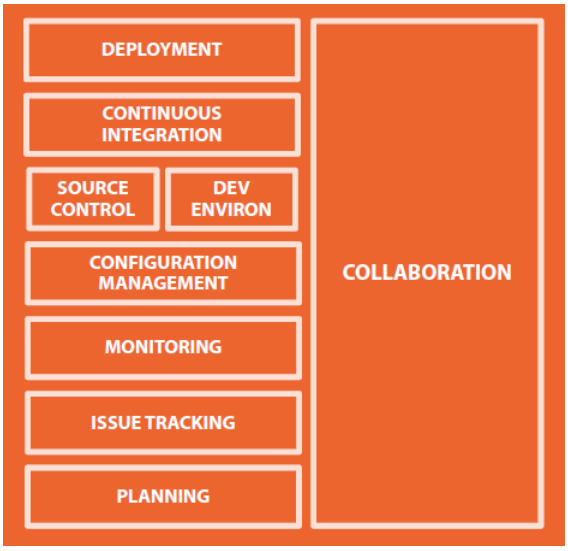
- Develops and verifies against production-like systems
- Reduces cost/time to deliver - Deploy often, deploy faster with repeatable, reliable process
- Increases Quality - Automated testing, Reduce cost/time to test
- Reduces Defect cycle time - Increase the ability to reproduce and fix defects
- Increases 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 (ELK)
  - New Relic
  - …..

Configuration Management

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

Source Control

- Controlled Assets
- Tools
  - Git
  - …

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 Feedback
Continuous Delivery
Continuous Testing
Continuous Deployment
Continuous Monitoring

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.

Jira Demo

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

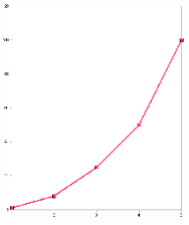
- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors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

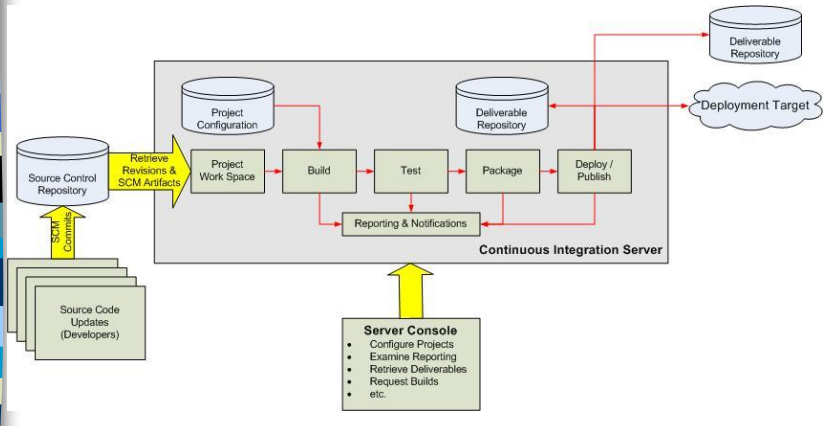
Continuous Integration Benefit

- 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



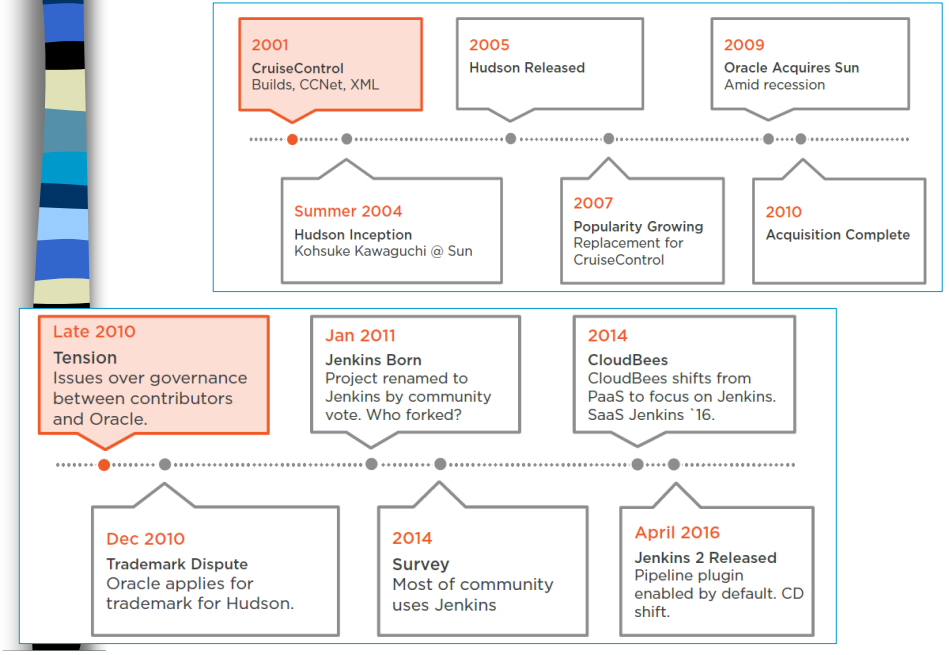
Ref: http://www.javaworld.com/javaworld/jw-12-2008/images/CIOverview.jpg

Continuous Integration Tools

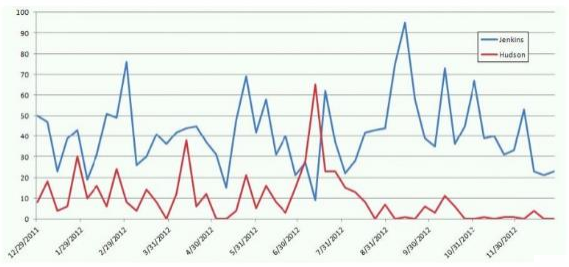
Name	Platform	License	Builders: Windows	Builders: Java	Builders: other	Notification	Integration, IDEs	Integration, other
Apache Gump	Python	Apache 2.0	Unknown	Ant, Maven 1	Unknown	Email	Unknown	Unknown
AppVeyor	Hosted, Self-Hosted	Proprietary	Visual Studio, MSBuild, Psake	No	Custom Script, PowerShell	Email, HipChat, Slack, Catlight	No	GitHub, Bitbucket, Kiln, Windows Azure
Azure DevOps Server (formerly TFS and VSTS)	Cross-platform	Proprietary, MIT	MSBuild, Visual Studio	Ant, Maven, Gradle, Android	C, C++, Go, Groovy, Java, Node.js, Perl, PHP, Python, Ruby	Email, SOAP, Catlight	Visual Studio, Eclipse, IntelliJ IDEA, Android Studio, Visual Studio Code	GitHub, Jenkins, Slack, Hipchat, FindBugs, Checkstyle, PMD
Bamboo	Web container	Proprietary	MSBuild,[2] NAnt,[3] Visual Studio[4]	Ant,[3] Maven 1-2-3[5]	Custom script, command-line tool, Bash, Xcode,[7] Phing,[8] Grunt, Grails[9]	XMPP, Google Talk, Email, RSS, Remote API, HipChat	IntelliJ IDEA, Eclipse, Visual Studio	FishEye, Jira, Clover, Bitbucket, GitHub
Buddy	Cross-platform	Proprietary	No	Ant, Maven, Gradle	Elixir, Go, Haskell, Node.js, PHP, Python, Ruby, .NET Core	Desktop, Email, Slack, SMS	No	Web Services, Bitbucket, GitHub, GitLab, Google Cloud Services, Heroku, Modulus
Buildbot	Python	GPL	Command-line	Command-line	Command-line	Email, Web, GUI, IRC	Unknown	Unknown
BuildMaster	Cross-platform	Proprietary	Yes	Yes	Cross-platform command-line	Email, custom	No	Many
GitLab	Hosted, Self-Hosted	Proprietary, MIT	Yes[10]	Maven,[11] Gradle[12]	SSH, Shell, VirtualBox, Parallels, Docker, Kubernetes, Custom[13]	Email, Web, Slack and others[14]	Gitpod,[15] WebIDE[16]	Many
GoCD	Cross-platform	Apache 2.0	Command-line	Command-line	Command-line	Email, hipchat, Slack, Gemit, Gitter, Riemann etc	No	GitHub
Jenkins	Web container	Creative Commons and MIT	MSBuild, NAnt, Batch Script	Ant, Maven 2, Kundo	CMake, Gant, Gradle, Grails, Phing, Rake, Ruby, SCons, Python, shell script, command-line	Android, Email, Google Calendar, IRC, XMPP, RSS, Twitter, Slack, Catlight, CCMenu, CCTray	Eclipse, IntelliJ IDEA, NetBeans	Bugzilla, Google Code, Jira, Bitbucket, Redmine, FindBugs, Checkstyle, PMD and Mantis, Trac, HP ALM
OpenMake Software Meister	Cross-platform	Proprietary	MSBuild, NAnt, Visual Studio	Ant, Maven 1-2-3	Shell script, batch script, cross-platform command-line, Groovy, Make, RTC Jazz, TFS Build, Custom Script Interpreter	Email, XMPP, RSS, Systray	Eclipse, Visual Studio	Bugzilla, Google Code, Jira, Bitbucket, Redmine, FindBugs, Checkstyle, PMD and Mantis, Trac

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

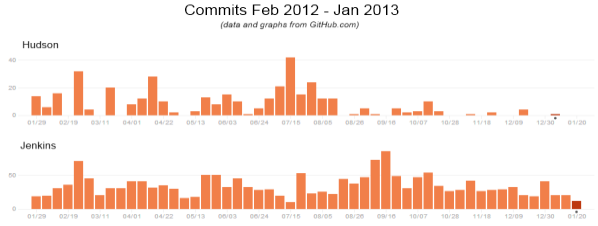
Before Jenkins

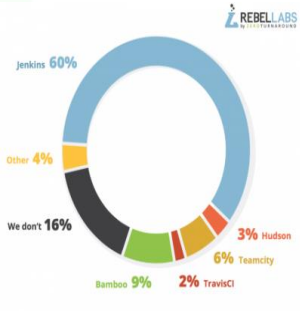


What Happened Next?



2012, THE DECISIVE YEAR

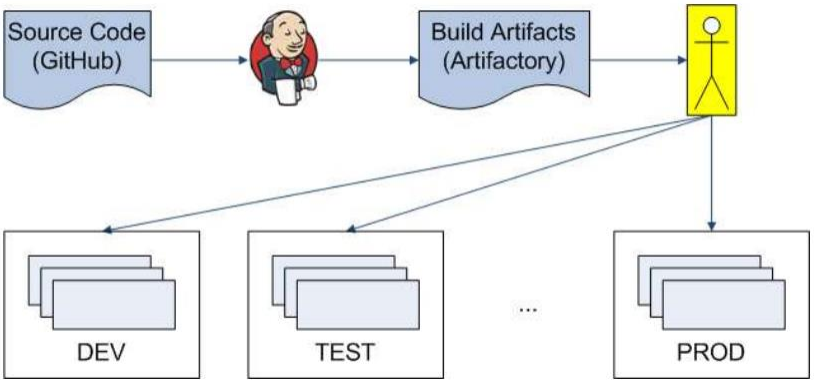




Standard Software Platform

- 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



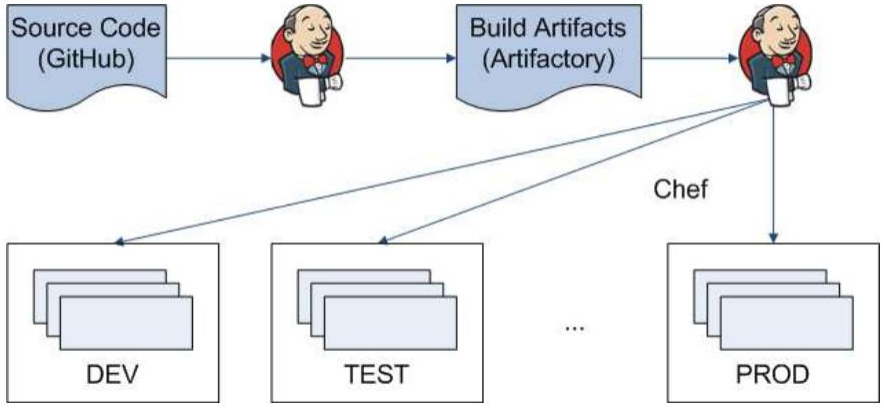
Initial Delivery Pipeline

- 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

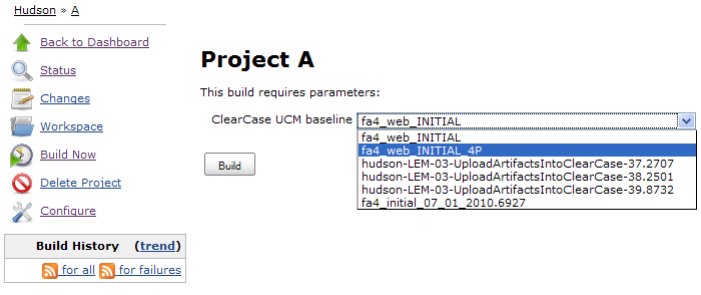
- Manually click build button
- Build periodically
- Build whenever a SNAPSHOT dependency is built
- Build after other projects are built
- Poll SCM
- IRC, Jabber, …

Get Source Code (1/2)

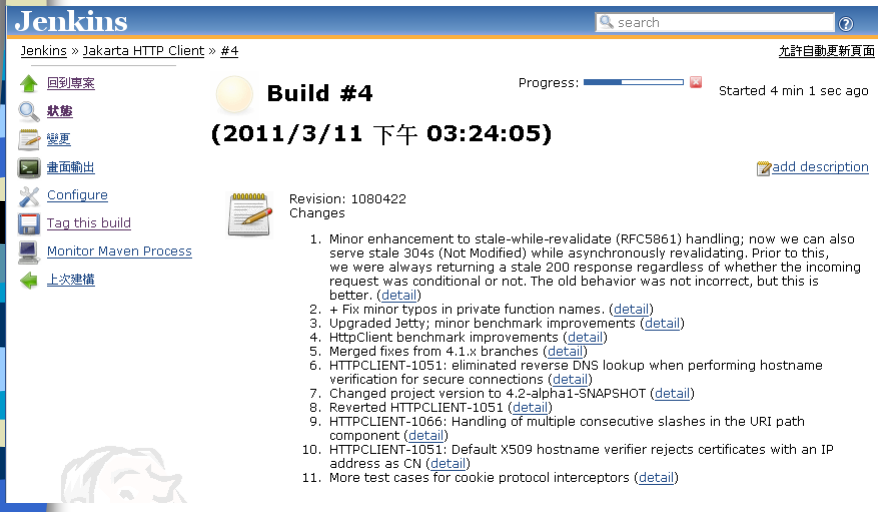
- CVS (build-in)
- SVN (build-in)
- GIT (requires Git)
- ClearCase (requires ClearCase)
- Mercurial, PVCS, VSS, …

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

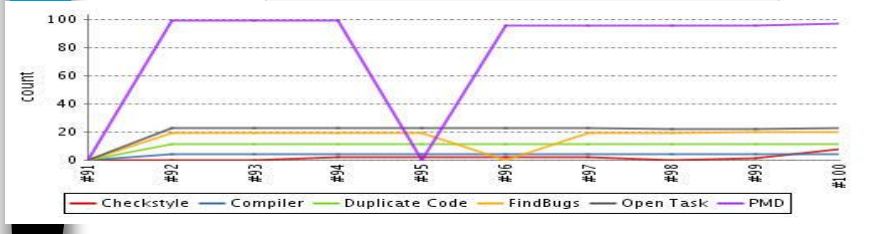
- Static Code Analysis
  - Checkstyle, PMD, Findbugs, Compiler Warning
- Test Report & Code Coverage
  - JUnit, TestNG, Cobertura, Clover
- Open Tasks

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	16	243	4	0	267
M-Single-Maven-Site	1784	2	72	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	789	
EmptyBlockCheck	29	
FinalClassCheck	5	
FinalParametersCheck	980	
HiddenFieldCheck	288	
HideUtilityClassConstructorCheck	9	
InnerAssignmentCheck	1	
InterfaceIsTypeCheck	8	
JavadocMethodCheck	1472	
JavadocPackageCheck	1	

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
16	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
16	0	10	6

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:137	19	BrowserCompatSpec.java:163
PlainSocketFactory.java:154	28	SSLSocketFactory.java:462
RequestTargetAuthentication.java:86	28	RequestProxyAuthentication.java:95
BrowserCompatSpec.java:163	19	NetscapeDraftSpec.java:137
DefaultRedirectHandler.java:141	44	DefaultRedirectStrategy.java:133
BrowserCompatSpec.java:128	34	BestMatchSpec.java:101
BasicDomainHandler.java:45	32	RFC2109DomainHandler.java:47
DefaultRedirectStrategy.java:133	44	DefaultRedirectHandler.java:141
BestMatchSpec.java:101	34	BrowserCompatSpec.java:128

Test Report

Tests

Job	Success		Failed		Skipped		Total
	#	%	#	%	#	%	#
Eden-ActiveWorlds	127	100%	0	0%	0	0%	127
Eden-Eden	266	100%	0	0%	0	0%	266
Eden-MondiDinamiciWebservice	155	100%	0	0%	0	0%	155
Eden-Palm	20	100%	0	0%	0	0%	20
Total	568	100%	0	0%	0	0%	568

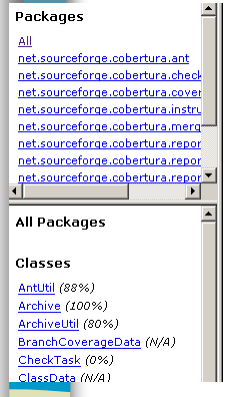
Test Result

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



Coverage Report - All Packages

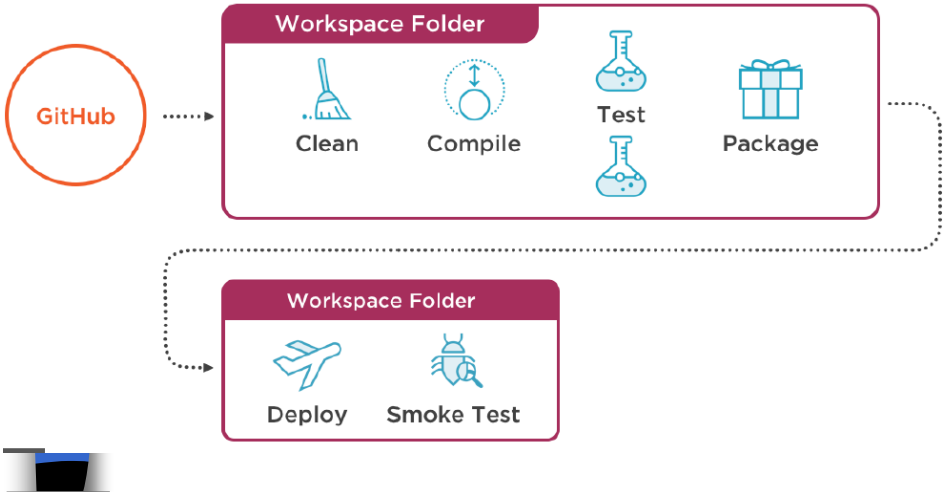
Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	55	75%	1625/2179	64%	472/738	2.319
net.sourceforge.cobertura.ant	11	52%	170/330	43%	40/94	1.848
net.sourceforge.cobertura.check	3	0%	0/150	0%	0/76	2.429
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	N/A	N/A	2.277
net.sourceforge.cobertura.instrument	10	90%	460/510	75%	123/164	1.854
net.sourceforge.cobertura.merge	1	86%	30/35	88%	14/16	5.5
net.sourceforge.cobertura.reporting	3	87%	116/134	80%	43/54	2.882
net.sourceforge.cobertura.reporting.html	4	91%	475/523	77%	156/202	4.444
net.sourceforge.cobertura.reporting.html.files	1	87%	39/45	62%	5/8	4.5
net.sourceforge.cobertura.reporting.xml	1	100%	155/155	95%	21/22	1.524
net.sourceforge.cobertura.util	9	60%	175/291	69%	70/102	2.892
someotherpackage	1	83%	5/6	N/A	N/A	1.2

Report generated by Cobertura 1.9 on 6/9/07 12:37 AM.

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

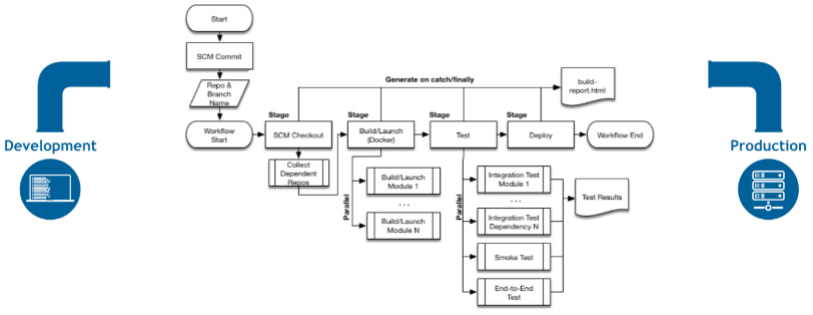
Declarative Vs Scripted Pipeline syntax

- 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

CI/CD Concepts Recap

- 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.

CI/CD Concepts Recap

- 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.

SonarQube

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

Selenium

What is Selenium?

- Selenium is an open-source automation testing tool that supports several scripting languages like Python, C#, Java, Perl, Ruby, JavaScript, etc., depending on the application to be tested. One can choose the script accordingly.

Components of Selenium

- Selenium WebDriver
  - The core component for browser automation
- Selenium IDE
  - A browser extension for recording and running tests
- Selenium Grid
  - Allows running tests on different machines and browsers in parallel
- Selenium RC (Remote Control)
  - Legacy component for executing tests

Supported Browsers

- Google Chrome
- Mozilla Firefox
- Internet Explorer/Edge
- Safari
- Opera

Key Features

- Cross-Browser Testing: Ability to test on multiple browsers.
- Parallel Execution: Running tests concurrently to reduce execution time.
- Integration: Compatibility with testing frameworks (JUnit, TestNG, NUnit) and CI/CD tools (Jenkins, GitLab CI).

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.

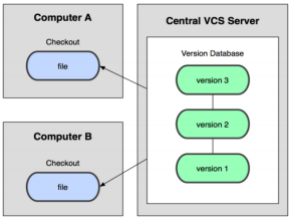
Version Control Systems

- 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?

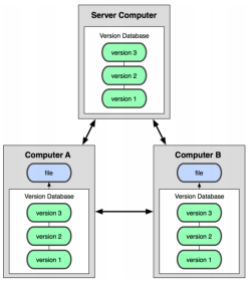
- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes

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

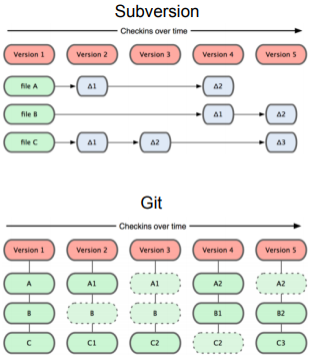
- Commit
  - Git thinks of its data like a set of snapshots of a mini file system.
  - Think of it as a save point during a video game.
- SHA
  - A SHA is basically an ID number for each commit.
  - Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6
- Branch
  - A branch is when a new line of development is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

Version Control Terminology

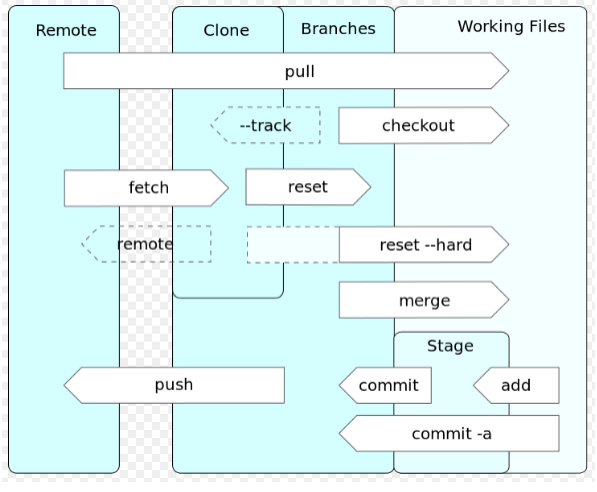
- Working Directory
  - files that you see in your computer's file system. When you open project files up on a code editor, you're working with files in the Working Directory.
- Checkout
  - Content in the repository has been copied to the Working Directory. Possible to checkout many things from a repository; 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.

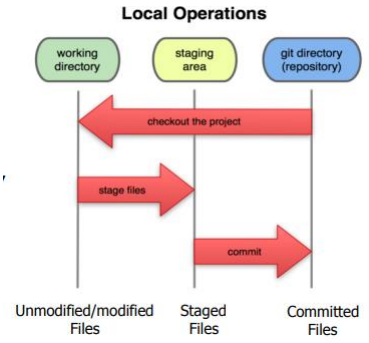


Git

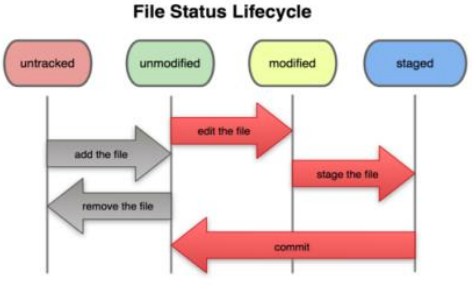


In your local copy on git, files can be:

- 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

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

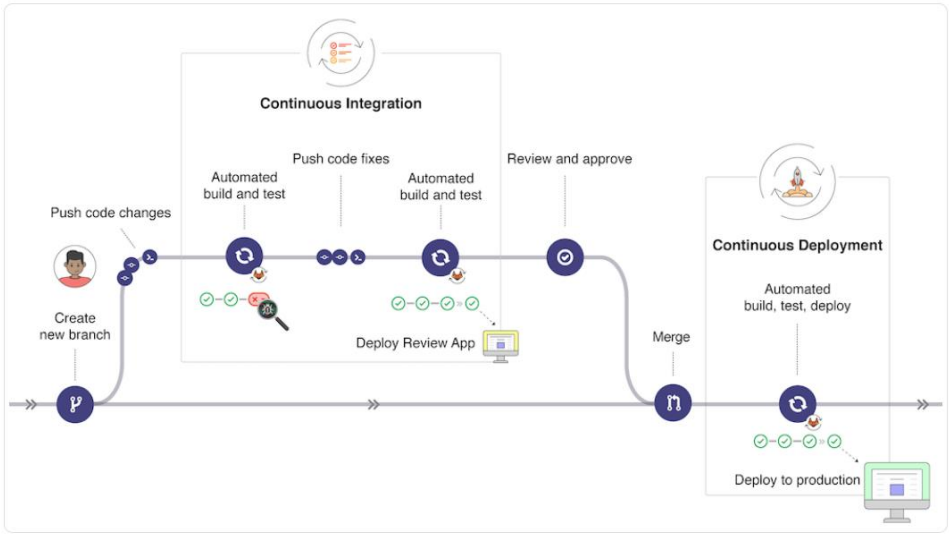
- Continuous Integration
  - For application with Git Repository code in GitLab, developers push code changes every hour/day.
  - For every push to the repository, create a set of scripts to build and test your application automatically.
  - These scripts help decrease the chances that you introduce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

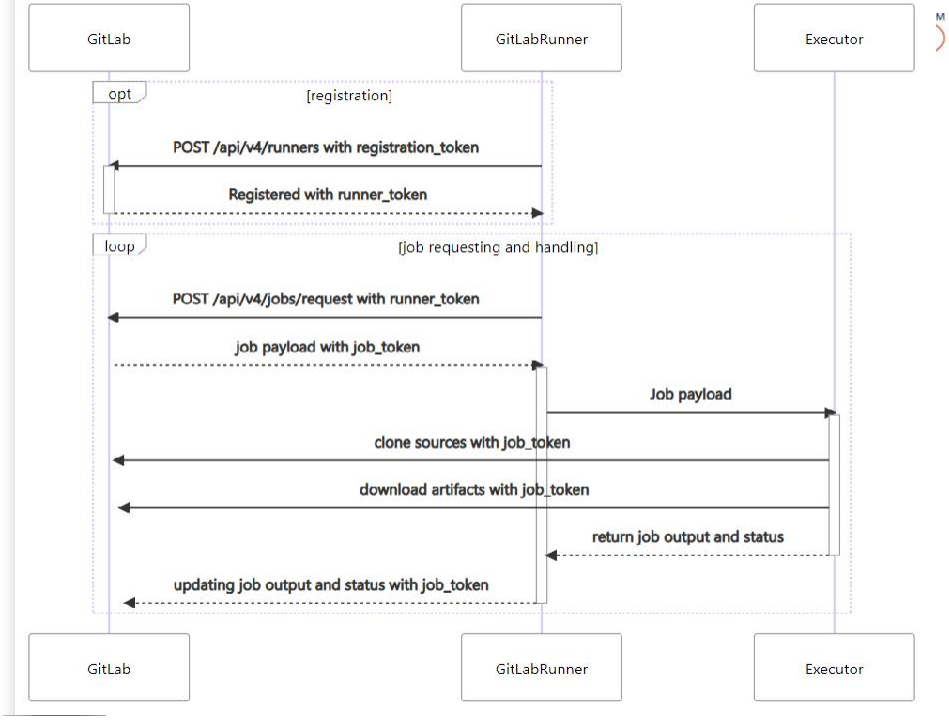
- Separate program that can be run on your local host, vm or even container
- Similar to jenkins Agent
- GitLab assigns pipeline jobs to available runners at runtime

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

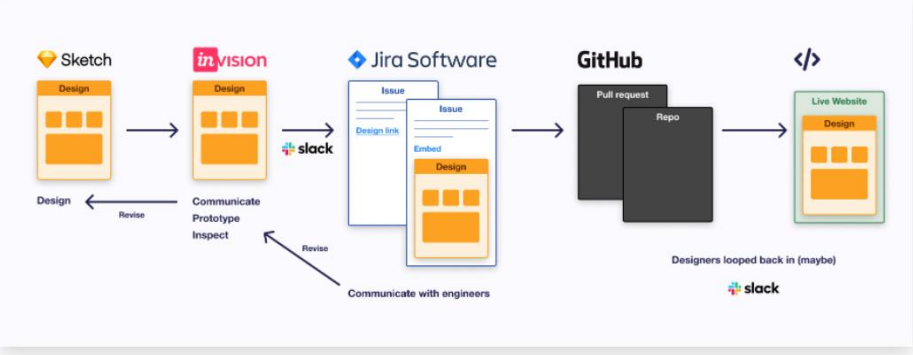
- Create pipeline using editor
- Pipeline
  - Jobs
  - Stages
  - Scripts
  - Artifacts
  - Variables
    - Masked
    - Protected

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)

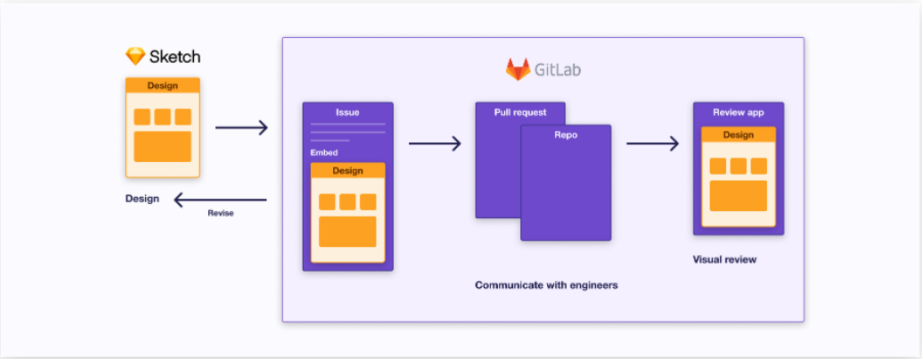
GitLab for Collaboration

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.

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

GitHub Actions

GitHub Actions

- GitHub Actions is a CI/CD (Continuous Integration and Continuous Deployment) and automation tool integrated directly into GitHub. It allows you to automate workflows, such as building, testing and deploying code as well as other tasks such as issue management and code review.

Key Components

- Workflows
  - A workflow is an automated process defined in a YAML file. It consists of one or more jobs that run in response to specific events (like a push or pull request).
- Jobs
  - Jobs are a set of steps that are executed on the same runner. They can run sequentially or in parallel.
- Steps
  - Steps are individual tasks within a job. Each step can execute commands or use actions.
- Hooks
  - hooks are scripts that run automatically every time a particular event occurs in a Git repository. They let you customize Git's internal behavior and trigger customizable actions at key points in the development life cycle.

Key Components

- Actions
  - Actions are reusable units of code that can be used in steps to perform various tasks like checking out code, setting up environments, or deploying applications
- Runners
  - Runners are servers that execute the jobs defined in workflows. GitHub provides hosted runners, but you can also set up self-hosted runners

Overview of CI/CD Concepts

- Continuous Integration (CI): Involves automatically building and testing code changes in a shared repository frequently. The goal is to detect issues early.
- Continuous Deployment (CD): Extends CI by automatically deploying code changes to production environments after passing tests. It ensures that the application is always in a deployable

Workflow Syntax and Structure

demo

Other Configurations

- Environment and Secrets
- Artifact Management
  - Storing and Retrieving Artifacts
  - Use actions/upload-artifact and actions/download-artifact
- Sharing Data Between Jobs
  - Save artifacts from one job and use in another

Variables

YAML

```yaml
name: Greeting on variable day

on:
  workflow_dispatch

env:
  DAY_OF_WEEK: Monday

jobs:
  greeting_job:
    runs-on: ubuntu-latest
    env:
      Greeting: Hello
    steps:
      - name: "Say Hello Mona it's Monday"
        run: echo "$Greeting $First_Name. Today is $DAY_OF_WEEK!"
        env:
          First_Name: Mona
```

Artifacts

```yaml
jobs:
  build_and_test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: npm install, build, and test
        run: |
          npm install
          npm run build --if-present
          npm test
      - name: Archive production artifacts
        uses: actions/upload-artifact@v4
        with:
          name: dist-without-markdown
          path: |
            dist
            !dist/**/*.md
      - name: Archive code coverage results
        uses: actions/upload-artifact@v4
        with:
          name: code-coverage-report
          path: output/test/code-coverage.html
```

Matrix Strategy

```yaml
strategy:
  matrix:
    os: [ubuntu-latest, windows-latest]
    node: [18, 20]
  exclude:
    - os: ubuntu-latest
      node: 18
```

- Enables you to run jobs across multiple versions of languages, operating systems, or other varying parameters. It's particularly useful for ensuring that your application works across different environments without needing to manually duplicate workflows.

Conditional Execution

- Prevent a job from running unless your conditions are met

```yaml
name: example-workflow
on: [push]
jobs:
  production-deploy:
    if: github.repository == 'octo-org/octo-repo-prod'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '14'
      - run: npm install -g bats
```

GitHub Vs GitLab

Parameters	GitLab	GitHub
Owned by	Gitlab Organization	Microsoft
Open-sourced	GitLab is open-source (for community edition)	GitHub is not open source.
Navigation	GitLab provides the feature of navigation into the repository.	GitHub allows users to navigate usability.
Project Analysis	GitLab provides user to see project development charts.	GitHub doesn't have this feature yet but they can check the commit history.
Security	More secure than Github, Extensive security features, including vulnerability management and compliance tools	It is less secure as security Dashboard, License Compliance is missing in GitHub.
Attachments	Gitlab supports adding other types of attachments.	GitHub does not allow adding other types of attachments.
CI/CD Integration	Built-in GitLab CI/CD with more extensive features	GitHub Actions for CI/CD pipelines
Issue Tracking	Advanced issue tracking; integrated project management tools	Basic issue tracking; Projects for task management

GitHub Vs GitLab

Advantages	GitLab is freely available and is open source for community edition It is a cloud-native application and is highly secure	It helps us create an organized document for the project. It is used for sharing the work in front of the public
Disadvantages	GitLab is available with many bugs and it makes the user experience sloppy. It is difficult to manage code reviews for first-timers	It is not open-source. It supports only Git version control

AWS Introduction

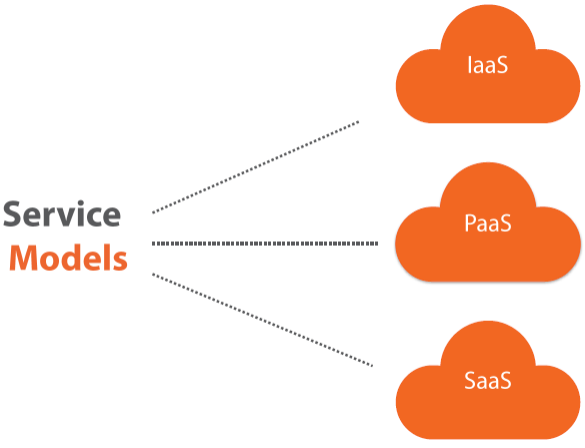
What is Cloud Computing?

- A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, server, storage, applications and services) that can be rapidly provisioned and released with minimal management effort of service provider interaction.

5 Essential Characters

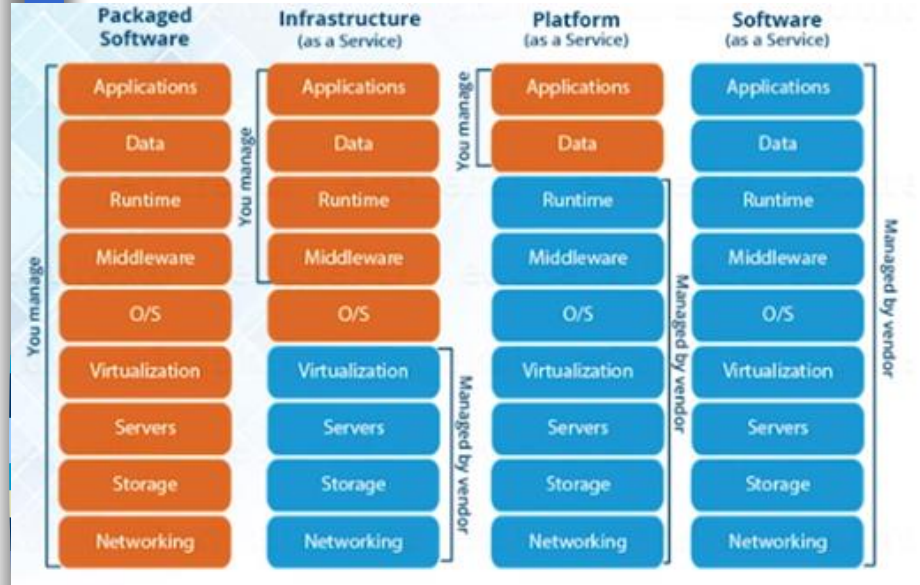
- On demand self services
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service
- Multi Tenacity

3 Service Models



AS A SERVICE

- IAAS: INFRASTRUCTURE
  - Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)
- PAAS: PLATFORM
  - Apprenda
- SAAS: SOFTWARE
  - Google Apps, Concur, Citrix GoToMeeting, Cisco WebEx



Cloud Compute Infrastructure

- Compute
- Storage
- Network
- Data Centers and Regions
- Shared Services
- Platforms

Who's who?

- Google Cloud Platform (GCP):
  - With only 5 years in operation, have created good presence in the market. The initial push was done to power their own services such as YouTube and Google. Later on, they built enterprise services and enabled anyone to host in the cloud.
- Amazon Web Services (AWS)
  - 11 years in operation, one of the oldest players in the cloud market. Their computing services are extensive and cover important cloud sections such as deployment, mobile networking, etc.
- Microsoft Azure
  - Azure is also 6 years old and has shown great promise in the market. They can easily be associated with the leader group in the market with AWS. Provides a complete set of cloud services.

AWS Global Infrastructure

- 16 Regions
  - "the western US, eastern US, central Europe"
- Each region consists of multiple availability zones
  - Separate data centers - Ireland is the region with 3 availability zones
  - Distinct locations from within an AWS region that are engineered to be isolated from failures (one can go down, others stay up)
- 54 Edge Locations
  - CDN end points - there are many more edge locations than regions
  - used by cloud front to cache files near the user where they access them to reduce latency

Services

- Storage
  - Simple Storage Service - S3
- Compute
  - Elastic Compute Cloud – EC2
- Database
  - Relational Database Service – RDS
- Security, Identity, & Compliance
  - Identity and Access Management - IAM

Lambda

Serverless on AWS

- Build & Run apps without thinking about servers
- Serverless is a way to describe the services, practices, and strategies that enable you to build more agile applications so you can innovate and respond to change faster.
- Infrastructure management tasks like capacity provisioning and patching are handled by AWS.
- Serverless services like AWS Lambda come with automatic scaling, built-in high availability, and a pay-for-value billing model.

Benefits

- Move from idea to market, faster
  - By eliminating operational overhead, your teams can release quickly, get feedback, and iterate to get to market faster.
- Lower your costs
  - pay-for-value billing, no need to over-provision
- Adapt at scale
  - Automatically scale from zero to peak demands,
- Build better applications, easier
  - Serverless applications have built-in service integrations, so you can focus on building your application instead of configuring it.

Serverless Services on AWS

- Compute
  - AWS Lambda
  - Amazon Fargate
- Data Store
  - Amazon S3
  - Amazon DynamoDB
  - Amazon RDS Proxy
  - Amazon Aurora Serverless
- Application Integration
  - Amazon EventBridge
  - AWS Step Functions
  - Amazon SQS
  - Amazon SNS
  - Amazon API Gateway
  - AWS AppSync

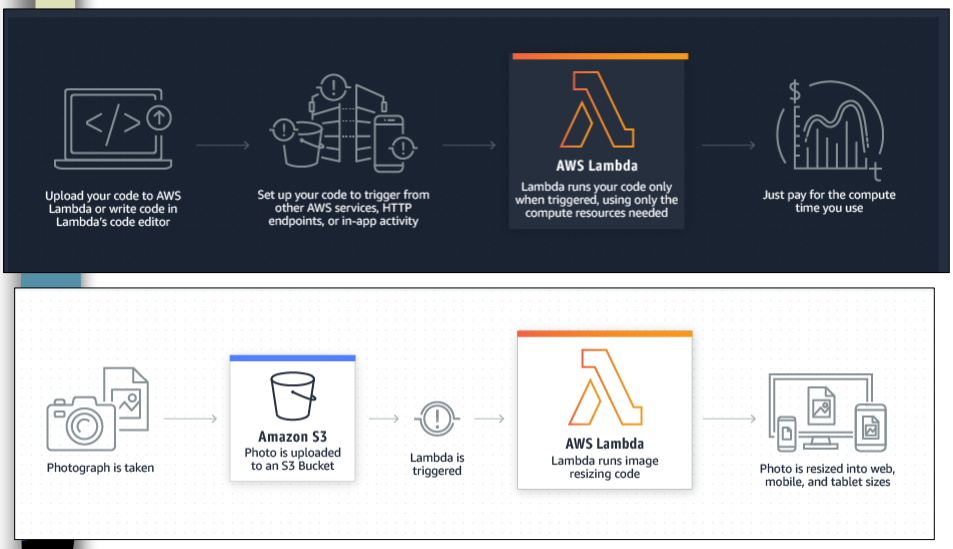
Lambda

- Run code without provisioning or managing servers and pay only for the resources you consume
- Benefits
  - No servers to manage
  - Continuous scaling
  - Cost optimized with millisecond metering
  - Consistent performance at any scale

Lambda

- Run Code for virtually any type of application or backend service - with zero administration
- Just upload your code as a ZIP file or container image, and Lambda will allocate compute execution power and run your code based on the incoming request or event, for any scale of traffic.
- You can set up your code to automatically trigger from over 200 AWS services and SaaS applications or call it directly from any web or mobile app.
- Language Support for Node.js, Python, Go, Java, and more and use both serverless and container tools, such as AWS SAM or Docker CLI, to build, test, and deploy your functions.

Lambda



What is a Lambda function?

- The code you run on AWS Lambda is called a “Lambda function.”
- After you create your Lambda function it is always ready to run as soon as it is triggered, similar to a formula in a spreadsheet.
- Each function includes your code as well as some associated configuration information, including the function name and resource requirements.

Lambda Functions

- Lambda functions are “stateless,” with no affinity to the underlying infrastructure, so that Lambda can rapidly launch as many copies of the function as needed to scale to the rate of incoming events.
- After you upload your code to AWS Lambda, you can associate your function with specific AWS resources (e.g. S3 bucket, Amazon DynamoDB table, Amazon Kinesis stream, or Amazon SNS notification).
- When the resource changes, Lambda will execute your function and manage the compute resources as needed in order to keep up with incoming requests.

Built-in Fault Tolerance

- AWS Lambda maintains compute capacity across multiple Availability Zones in each region to help protect your code against individual machine or data center facility failures.
- Both AWS Lambda and the functions running on the service provide predictable and reliable operational performance.
- AWS Lambda is designed to provide high availability for both the service itself and for the functions it operates. No maintenance windows or scheduled downtimes.

Lambda Concepts

- Function
- Trigger
- Event
- Execution environment
- Deployment package
- Runtime
- Layer
- Extension
- Concurrency
- Qualifier

Lambda Concepts

- Function
  - A function is a resource that you can invoke to run your code in Lambda. A function has code to process the events that you pass into the function or that other AWS services send to the function.
- Trigger
  - A trigger is a resource or configuration that invokes a Lambda function. This includes AWS services that you can configure to invoke a function, applications that you develop, and event source mappings. An event source mapping is a resource in Lambda that reads items from a stream or queue and invokes a function

Lambda Concepts

- Event
  - An event is a JSON-formatted document that contains data for a Lambda function to process. The runtime converts the event to an object and passes it to your function code. When you invoke a function, you determine the structure and contents of the event.
- Execution environment
  - An execution environment provides a secure and isolated runtime environment for your Lambda function. An execution environment manages the processes and resources that are required to run the function. The execution environment provides lifecycle support for the function and for any extensions associated with your function.

Lambda Concepts

- Deployment package
  - You deploy your Lambda function code using a deployment package. Lambda supports two types of deployment packages:
    - A .zip file archive that contains your function code and its dependencies. Lambda provides the operating system and runtime for your function.
    - A container image that is compatible with the Open Container Initiative (OCI) specification. You add your function code and dependencies to the image. You must also include the operating system and a Lambda runtime.

Lambda Concepts

- Runtime
  - The runtime provides a language-specific environment that runs in an execution environment. The runtime relays invocation events, context information, and responses between Lambda and the function. You can use runtimes that Lambda provides, or build your own. If you package your code as a .zip file archive, you must configure your function to use a runtime that matches your programming language. For a container image, you include the runtime when you build the image.
- Layer
  - A Lambda layer is a .zip file archive that can contain additional code or other content. A layer can contain libraries, a custom runtime, data, or configuration files.

Lambda Concepts

- Extension
  - Lambda extensions enable you to augment your functions. For example, you can use extensions to integrate your functions with your preferred monitoring, observability, security, and governance tools.
- Concurrency
  - Concurrency is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is provisioned, increasing the function's concurrency.

Lambda Concepts

- Qualifier
  - When you invoke or view a function, you can include a qualifier to specify a version or alias. A version is an immutable snapshot of a function's code and configuration that has a numerical qualifier. For example, my-function:1. An alias is a pointer to a version that you can update to map to a different version, or split traffic between two versions. For example, my-function:BLUE. You can use versions and aliases together to provide a stable interface for clients to invoke your function.

Context

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:",
context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time
remaining in get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:",
context.get_remaining_time_in_millis())
```

Logging

- Base Logging
  - print()
- Logging Library

```
import os
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ) logger.info('## EVENT')
    logger.info(event)
```

Versioning

- Use versions to manage deployment of your functions
- A function version includes the following information:
  - The function code and all associated dependencies.
  - The Lambda runtime that invokes the function.
  - All of the function settings, including the environment variables.
  - A unique Amazon Resource Name (ARN) to identify the specific version of the function.

Invocations

- Invoke using Lambda console, the Lambda API, the AWS SDK, the AWS CLI, and AWS toolkits.
- You can also configure other AWS services to invoke your function, or you can configure Lambda to read from a stream or queue and invoke your function.
- When you invoke a function, you can choose to invoke it synchronously or asynchronously.

Synchronous Vs Asynchronous

- Synchronous invocation
  - Wait for the function to process the event and return a response.
- Asynchronous invocation
  - Lambda queues the event for processing and returns a response immediately. For asynchronous invocation, Lambda handles retries and can send invocation records to a destination.

AWS CLI

- Synchronous
  - aws lambda invoke --function-name my-function --payload '{ "key": "value" }' response.json
- Asynchronous
  - aws lambda invoke --function-name my-function --invocation-type Event --payload '{ "key": "value" }' response.json

Concurrency for a Lambda function

- Two types of Concurrency Controls:
  - Reserved concurrency – Guarantees the maximum number of concurrent instances for the function. When a function has reserved concurrency, no other function can use that concurrency. There is no charge for configuring reserved concurrency for a function.
  - Provisioned concurrency –Initializes a requested number of execution environments so that they are prepared to respond immediately to your function's invocations. Note that configuring provisioned concurrency incurs charges to your AWS account.

Cold Vs Hot Container

- When running a serverless function, it will stay active (a.k.a., hot) as long as you're running it. Your container stays alive, ready and waiting for execution.
- After a period of inactivity, your cloud provider will drop the container, and your function will become inactive, (a.k.a., cold).
- A cold start happens when you execute an inactive function. The delay comes from your cloud provider provisioning your selected runtime container and then running your fn.

Temporary storage with /tmp

- The Lambda execution environment provides a file system for your code to use at /tmp. This space has a fixed size of 512 MB. The same Lambda execution environment may be reused by multiple Lambda invocations to optimize performance. The /tmp area is preserved for the lifetime of the execution environment and provides a transient cache for data between invocations. Each time a new execution environment is created, this area is deleted.

```
import os, zipfile
os.chdir('/tmp')
with zipfile.ZipFile(myzipfile, 'r') as zip:
    zip.extractall()
```

API Gateway

- Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
- APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services.
- Create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications.
- API Gateway supports containerized and serverless workloads & web applications.

Benefits of API Gateway

- Efficient API development
- Easy Monitoring
- Performance at any scale
- Flexible security controls
- Cost savings at scale
- RESTful API options

Exposing Lambda function as API Gateway

Step Functions

Step Functions

- Serverless function orchestrator that makes it easy to sequence AWS Lambda functions and multiple AWS services into business-critical applications.
- Through its visual interface, you can create and run a series of checkpointed and event-driven workflows that maintain the application state. The output of one step acts as an input to the next.
- Each step in your application executes in order, as defined by your business logic.

Main Features

- Sequencing
- Error Handling
- Retry Logic
- Branching
- Parallel Paths
- Human Interaction
- Retaining state across function calls
- Tracing

Workflow Types

- Step Functions has two workflow types.
- Standard workflows
  - have exactly-once workflow execution and can run for up to one year.
  - Ideal for long-running, auditable workflows, as they show execution history and visual debugging
- Express workflows
  - have at-least-once workflow execution and can run for up to five minutes.
  - Ideal for high-event-rate workloads, such as streaming data processing and IoT data ingestion.

Standard and Express workflows

- Standard workflows
  - 2,000 per second execution rate
  - 4,000 per second state transition rate
  - Priced per state transition
  - Shows execution history and visual debugging
  - Supports all service integrations and patterns
- Express workflows
  - 100,000 per second execution rate
  - Nearly unlimited state transition rate
  - Priced per number and duration of executions
  - Sends execution history to CloudWatch
  - Supports all service integrations and most patterns

Use Cases

- Use case #1: Function orchestration
- Use case #2: Branching
- Use case #3: Error handling
- Use case #4: Human in the loop
- Use case #5: Parallel processing
- Use case #6: Dynamic parallelism

Service Integrations

- Request a response (default)
  - Call a service, and let Step Functions progress to the next state after it gets an HTTP response.
- Run a job (.sync)
  - Call a service, and have Step Functions wait for a job to complete.
- Wait for a callback with a task token (.waitForTaskToken)
  - Call a service with a task token, and have Step Functions wait until the task token returns with a callback.

States

- Individual states can make decisions based on their input, perform actions, and pass output to other states.
- In AWS Step Functions you define your workflows in the Amazon States Language.
- The Step Functions console provides a graphical representation of that state machine to help visualize your application logic.
- States are elements in your state machine. A state is referred to by its name, which can be any string, but which must be unique within the scope of the entire state machine.

State Functions

- Task
  - Do some work in your state machine (a Task state)
- Choice
  - Make a choice between branches of execution
- Fail or Succeed
  - Stop an execution with a failure or success
- Pass
  - Simply pass its input to its output or inject some fixed data
- Wait
  - Provide a delay for a certain amount of time or until a specified time/date
- Parallel
  - Begin parallel branches of execution
- Map
  - Dynamically iterate steps

Pass State

```
{
  "Comment": "A Hello World example for  Pass states",
  "StartAt": "Hello",
 "States": {
    "Hello": {
      "Type": "Pass",
      "Result": "Hello",
      "Next": "World"
    },
    "World": {
      "Type": "Pass",
      "Result": "World",
      "End": true } } }
```

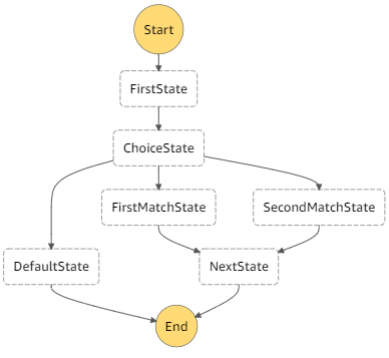
Map – Array Processing

```
{
  "Comment": "..",
  "StartAt": "Map",
  "States": {
    "Map": {
      "Type": "Map",
      "ItemsPath": "$.array",
      "ResultPath": "$.array",
      "MaxConcurrency": 2,
      "Next": "Final State",
      "Iterator": {
        "StartAt": "Pass",
}
```

```
"States": {
        "Pass": {
          "Type": "Pass",
          "Result": "Done!",
          "End": true
        }
      }
    }
  },
  "Final State": {
    "Type": "Pass",
    "End": true
  }
}
```

Choice - Branching

```
"ChoiceState": {
   "Type" : "Choice",
   "Choices": [
   {
    "Variable": "$.foo",
    "NumericEquals": 1,
    "Next": "FirstMatchState"
    },
    {
    "Variable": "$.foo",
    "NumericEquals": 2,
    "Next": "SecondMatchState"
     }
    ],
    "Default": "DefaultState"
   }
```



Wait State – Timed wait

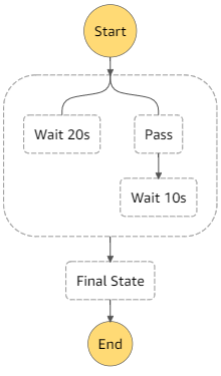
```
"wait_using_seconds": {
      "Type": "Wait",
      "Seconds": 10,
    },

"wait_using_timestamp": {
      "Type": "Wait",
      "Timestamp": "2015-09-04T01:59:00Z",
    },
```

```
"wait_using_timestamp_path": {
      "Type": "Wait",
      "TimestampPath": "$.expirydate",
    },
"wait_using_seconds_path": {
      "Type": "Wait",
      "SecondsPath": "$.expiryseconds",
    },
```

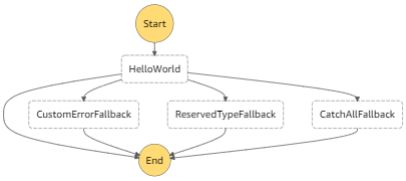
Parallel – Parallel Branches

```
"Parallel": {
     "Type": "Parallel",
     "Next": "Final State",
     "Branches": [
      { "StartAt": "Wait 20s",
        "States": { "Wait 20s": {
            "Type": "Wait",
            "Seconds": 20,
            "End": true      } } },
      { "StartAt": "Pass",
        "States": {
          "Pass": {
            "Type": "Pass",
            "Next": "Wait 10s“     },
          "Wait 10s": {
            "Type": "Wait",
            "Seconds": 10,
            "End": true            }
```



Catch – Handling Errors

```
"HelloWorld": {
 "Type": "Task",
 "Resource": "lambdaArn",
 "Catch": [     {
 "ErrorEquals":["CustomError"],
 "Next": "CustomErrorFallback“
},
 {
"ErrorEquals":["States.TaskFailed"],
 "Next": ReservedTypeFallback"
},
 {    "ErrorEquals": ["States.ALL"],
       "Next": "CatchAllFallback“ }
    ],
      "End": true
    },
```



Periodically Starting

- CloudWatch Events
  - Delivers a near real-time stream of system events that describe changes in AWS resources
- CloudWatch Events
  - Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console

Periodically Starting

- Start state machine execution after every one minute
  - Create State Machine
  - Select State machine
    - Actions
    - Create event bridge(CloudWatch) rule
    - After every 1 minute
  - Check
    - State Machine Executions

Exposing as API Gateway

- You can use Amazon API Gateway to associate your AWS Step Functions APIs with methods in an API Gateway API. When an HTTPS request is sent to an API method, API Gateway invokes your Step Functions API actions.
- Start a Step Functions execution by calling StartExecution and DescribeExecution to get the result.

Amazon S3 Events as trigger

- Amazon EventBridge to execute an AWS Step Functions state machine in response to an event or on a schedule.

Activities

- Activities are an AWS Step Functions feature that enables you to have a task in your state machine where the work is performed by a worker that can be hosted on Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Container Service (Amazon ECS), AWS Lambda, mobile devices—basically anywhere.

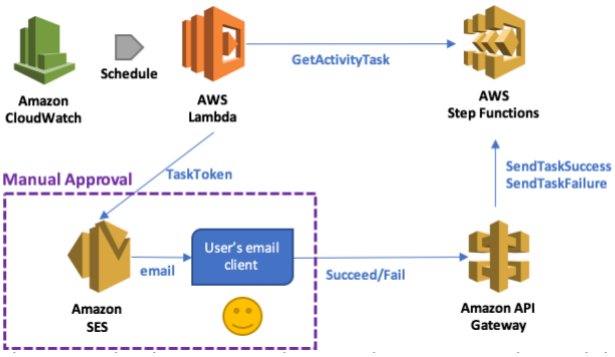
Overview

- In AWS Step Functions, activities are a way to associate code running somewhere (known as an activity worker) with a specific task in a state machine.
- You can create an activity using the Step Functions console, or by calling CreateActivity. This provides an Amazon Resource Name (ARN) for your task state. Use this ARN to poll the task state for work in your activity worker.

Human Interaction

- The most important feature of creating workflows with Step Functions is the ability to control its execution with external input. We can pause, continue, or stop the workflows whenever we want. This is especially important when we need to have human interactions with the workflow.

Serverless Manual Approval



https://aws.amazon.com/blogs/compute/implementing-serverless-manual-approval-steps-in-aws-step-functions-and-amazon-api-gateway/

Maintaining

- Logging and monitoring
  - Cloud Watch
    - Metrics
    - Alarms
  - EventBridge Events
  - AWS CloudTrail
- Logging using CloudWatch Logs
  - X-Ray

X-Ray

- Use AWS X-Ray to visualize the components of your state machine, identify performance bottlenecks, and troubleshoot requests that resulted in an error.
- State machine sends trace data to X-Ray, and X-Ray processes the data to generate a service map and searchable trace summaries.

X-Ray

- This gives you a detailed overview of an entire Step Functions request. Step Functions will send traces to X-Ray for state machine executions, even when a trace ID is not passed by an upstream service. You can use an X-Ray service map to view the latency of a request, including any AWS services that are integrated with X-Ray. You can also configure sampling rules to tell X-Ray which requests to record, and at what sampling rates, according to criteria that you specify.

Cloud Formation

Refer to CloudFormation Pdf

Monte Carlo

Monte Carlo

- A Monte Carlo simulation is a model used to predict the probability of a variety of outcomes when the potential for random variables is present.
- Monte Carlo simulations help to explain the impact of risk and uncertainty in prediction and forecasting models.

Monte Carlo

- A Monte Carlo simulation requires assigning multiple values to an uncertain variable to achieve multiple results and then averaging the results to obtain an estimate.
- These simulations assume perfectly efficient markets.
- Monte Carlo simulations are increasingly used in conjunction with artificial intelligence.

DataDog

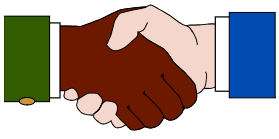
DataDog

- Datadog is the cloud-native monitoring and security platform for infrastructure, applications, logs, and more.
- Datadog is an observability platform that supports every phase of software development on any stack. The platform consists of many products that help you build, test, monitor, debug, optimize, and secure your software. These products can be used individually or combined into a customized solution

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com