Jenkins

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.
www.fandsindia.com

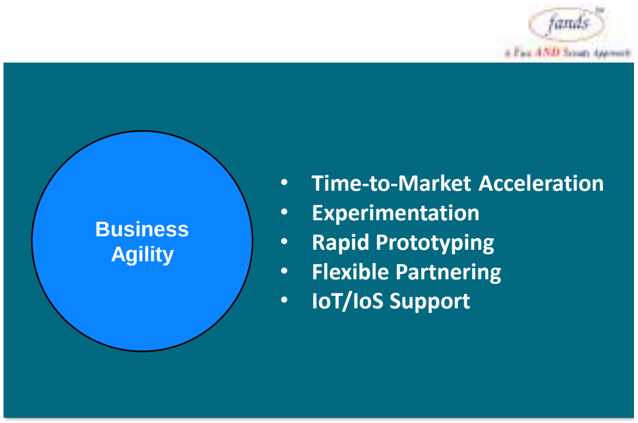
Ground Rules

- 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

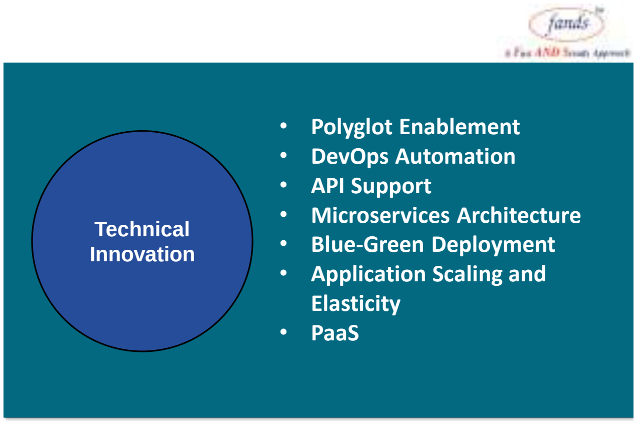
Discuss

Why Change?



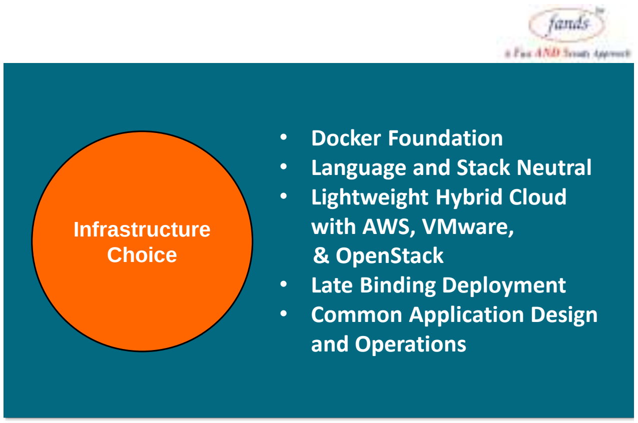
Business Agility

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



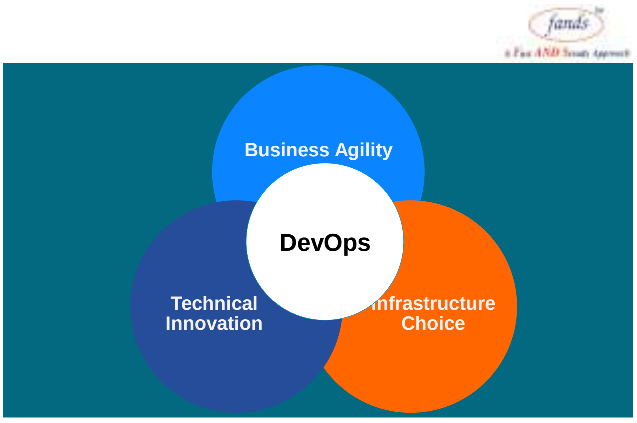
Technical Innovation

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

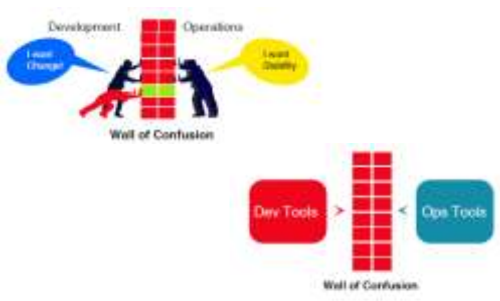


Infrastructure Choice

- Docker Foundation
- Language and Stack Neutral
- Lightweight Hybrid Cloud with AWS, VMware, & OpenStack
- Late Binding Deployment
- Common Application Design and Operations



Dev-Ops



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

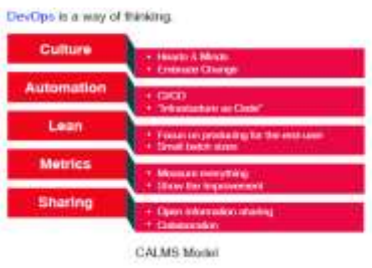
DevOps Is Fixing It

- 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.

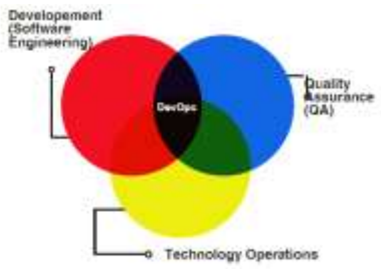
Why DevOps?

- 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



Five Basic Principles of DevOps

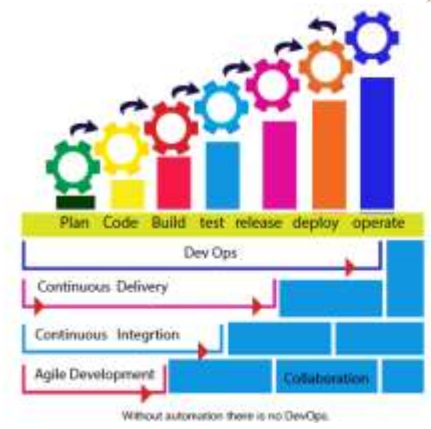
- Eliminate the blame game, Open post-mortems, Feedback, Rewarding failures
- Continous Delivery, Monitoring, Configuration Management
- Business value for end user
- Performance Metrics, Logs, Business goals Metrics,
- People Integration Metrics, KPI
- Ideas, Plans, Goals, Metrics, Complications, Tools

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



Monoliths to Micro Services

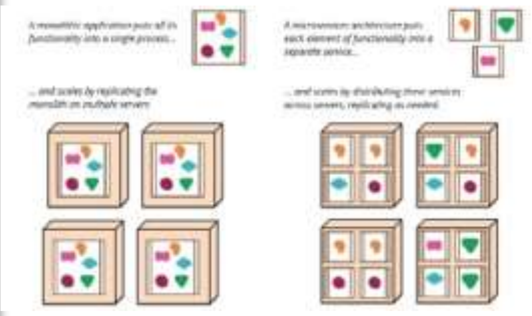
Monoliths to Micro Services

- 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


A monolithic application puts all its functionality into a single process…

A microservices architecture puts each element of functionality into a separate service…

… and scales by replicating the monolith on multiple servers

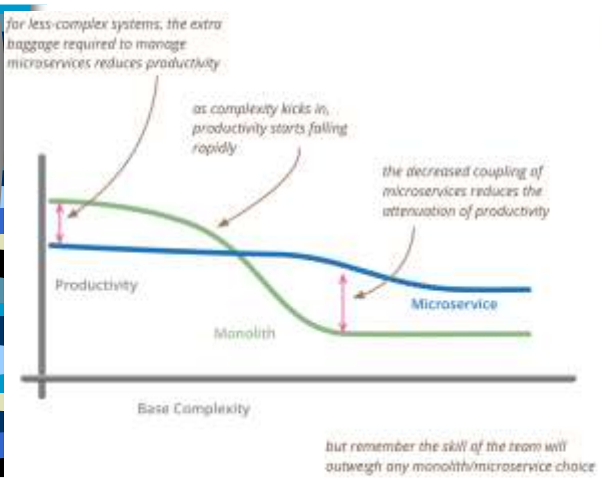
… and scales by distributing these services across servers, replicating as needed.

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

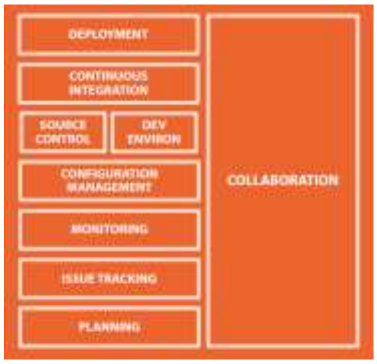


Issues with Micro Services

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

Back to DevOps

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
  - ..

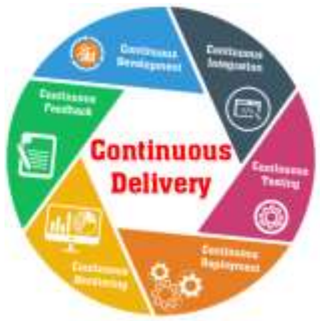
Continuous Integration

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

Deployment

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

Continuous…



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.*

Continuous Integration

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

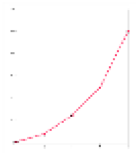
- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors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

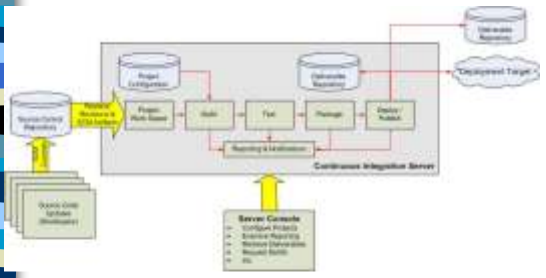
Continuous Integration Benefit

- 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



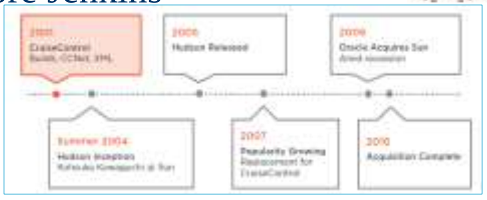
Ref: http://www.javaworld.com/javaworld/jw-12-2008/images/CIOverview.jpg

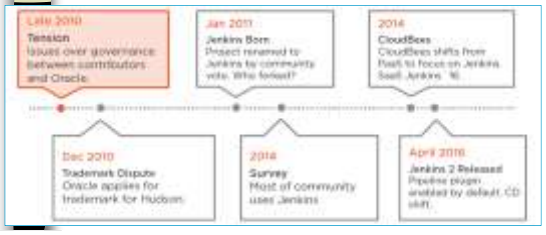
Continuous Integration Tools

[illegible]

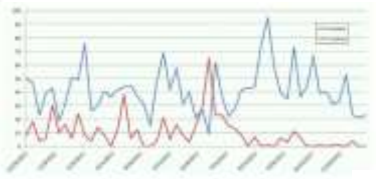
Ref: http://en.wikipedia.org/wiki/Comparison_of_Continuous_Integration_Software

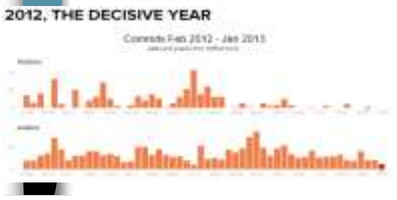
Before Jenkins

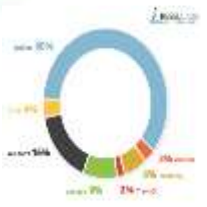



Late 2010
Tension
Issues over governance between contributors and Oracle
Dec 2010
Trademark Dispute
Oracle applies for trademark for Hudson
Jan 2011
Jenkins Born
Project renamed to Jenkins by community vote. Who forked?
2014
Survey
Most of community uses Jenkins
2014
CloudBees
CloudBees shifts from PaaS to focus on Jenkins
April 2016
Jenkins 2 Released
Pipeline plugin enabled by default. CD shift.

What Happened Next?





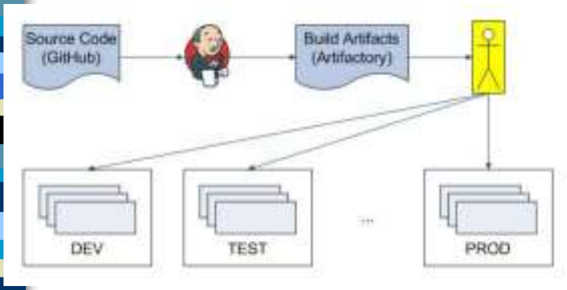


Jenkins

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



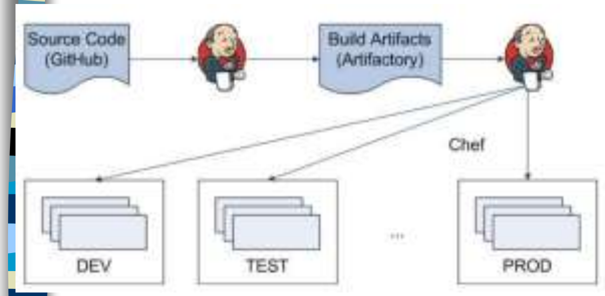
Initial Delivery Pipeline

- 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

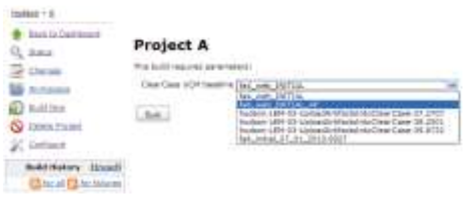
- Manually click build button
- Build periodically
- Build whenever a SNAPSHOT dependency is built
- Build after other projects are built
- Poll SCM
- IRC, Jabber, …

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)

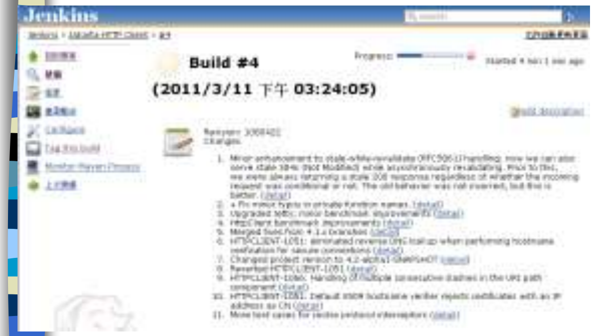
- Get current snapshot
- Get baseline (tag)



This build requires parameters:

ClearCase UCM baseline [illegible]

Code Change History



Build #4

(2011/3/11 下午 03:24:05)

Revision: 1080421 [illegible]
Changes

1. Minor enhancement to stale-while-revalidate (RFC5861) handling: now we can also serve stale 304s (Not Modified) while asynchronously revalidating. Prior to this, we were always returning a stale 200 response regardless of whether the incoming request was conditional or not. The old behavior was not incorrect, but this is better. (detail)
2. + Fix minor typos in private function names. (detail)
3. Upgraded [illegible] minor benchmark improvements (detail)
4. HttpClient benchmark improvements (detail)
5. Merged fixes from 4.1.x branches (detail)
6. HTTPCLIENT-1051: eliminated reverse DNS lookup when performing hostname verification for secure connections (detail)
7. Changed project version to 4.2-alpha1-SNAPSHOT (detail)
8. Reverted HTTPCLIENT-1051 (detail)
9. HTTPCLIENT-1060: Handling of multiple consecutive slashes in the URI path component (detail)
10. HTTPCLIENT-1061: Default X509 hostname verifier rejects certificates with an IP address as CN (detail)
11. More test cases for revalidation interceptors (detail)

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
- …

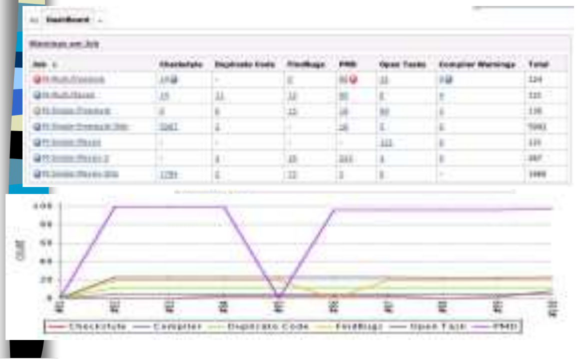
Build Notifier

- 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

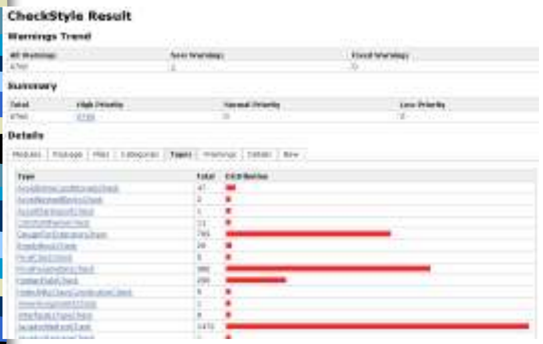
Static Code Analysis



Dashboard

Job	Checkstyle	Duplicate Code	FindBugs	PMD	Open Tasks	Compiler Warnings	Total
[illegible]	[illegible]	-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	-	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	-	-	-	-	[illegible]	[illegible]	[illegible]
[illegible]	-	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-	[illegible]

CheckStyle



CheckStyle Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
[illegible]	[illegible]	[illegible]

Summary

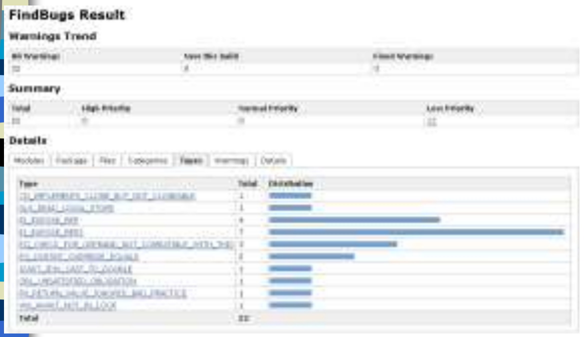
Total	High Priority	Normal Priority	Low Priority
[illegible]	[illegible]	[illegible]	[illegible]

Details

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

Type	Total	Distribution
[illegible]	47	
[illegible]	2	
[illegible]	1	
[illegible]	11	
[illegible]	765	
[illegible]	28	
[illegible]	5	
[illegible]	990	
[illegible]	290	
[illegible]	6	
[illegible]	1	
[illegible]	8	
[illegible]	1470	

FindBugs



FindBugs Result

Warnings Trend

All Warnings	New this build	Fixed Warnings
22	[illegible]	[illegible]

Summary

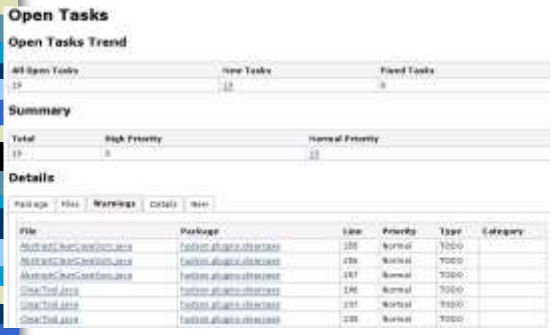
Total	High Priority	Normal Priority	Low Priority
22	[illegible]	[illegible]	22

Details

Modules | Packages | Files | Categories | Types | Warnings | Details

Type	Total	Distribution
[illegible]	1	
[illegible]	1	
[illegible]	4	
[illegible]	7	
[illegible]	3	
[illegible]	2	
[illegible]	1	
[illegible]	1	
[illegible]	1	
[illegible]	1	
Total	22	

Open Tag



Open Tasks

Open Tasks Trend

All Open Tasks	New Tasks	Fixed Tasks
19	19	0

Summary

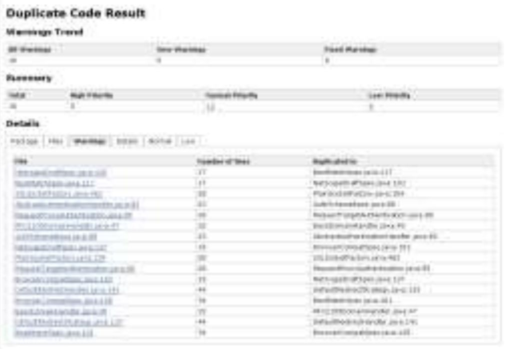
Total	High Priority	Normal Priority
19	0	19

Details

Package | Files | Warnings | Details | New

File	Package	Line	Priority	Type	Category
[illegible]	[illegible]	[illegible]	Normal	TODO	
[illegible]	[illegible]	[illegible]	Normal	TODO	
[illegible]	[illegible]	[illegible]	Normal	TODO	
[illegible]	[illegible]	[illegible]	Normal	TODO	
[illegible]	[illegible]	[illegible]	Normal	TODO	
[illegible]	[illegible]	[illegible]	Normal	TODO	

Duplicate Code



Duplicate Code Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
[illegible]	[illegible]	[illegible]

Summary

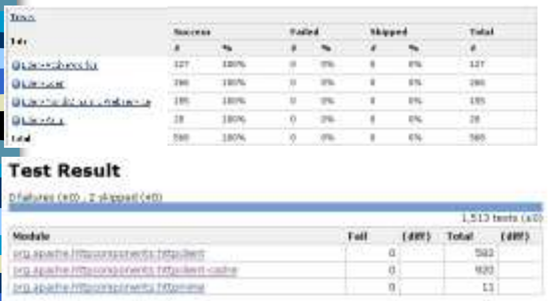
Total	High Priority	Normal Priority	Low Priority
[illegible]	[illegible]	[illegible]	[illegible]

Details

Package | Files | Warnings | [illegible] | [illegible] | [illegible]

File	Number of lines	Duplicated in
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	44	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	44	[illegible]
[illegible]	[illegible]	[illegible]

Test Report



Tests

Title	Success #	Success %	Failed #	Failed %	Skipped #	Skipped %	Total #
[illegible]	127	100%	0	0%	0	0%	127
[illegible]	286	100%	0	0%	0	0%	286
[illegible]	185	100%	0	0%	0	0%	185
[illegible]	28	100%	0	0%	0	0%	28
Total	566	100%	0	0%	0	0%	566

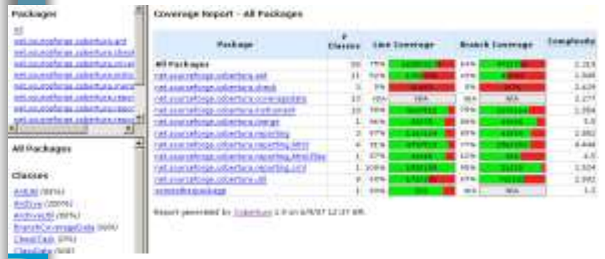
Test Result

0 failures (±0) , 2 skipped (±0)

1,513 tests (±0)

Module	Fail	(diff)	Total	(diff)
org.apache.httpcomponents:httpclient	0		583	
org.apache.httpcomponents:httpclient-cache	0		920	
org.apache.httpcomponents:httpmime	0		11	

Test Code Coverage



Packages

All Packages

Classes

AntUtil (88%)
Archive (100%)
ArchiveUtil (80%)
BranchCoverageData (N/A)
CheckTask (0%)
ClassData (N/A)

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	55	75%	64%	2.319
net.sourceforge.cobertura.ant	11	52%	47%	1.548
net.sourceforge.cobertura.check	3	0%	0%	2.429
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	2.277
net.sourceforge.cobertura.instrument	10	90%	75%	1.954
net.sourceforge.cobertura.merge	1	86%	88%	5.0
net.sourceforge.cobertura.reporting	2	87%	87%	2.882
net.sourceforge.cobertura.reporting.html	4	91%	77%	4.444
net.sourceforge.cobertura.reporting.html.files	1	87%	12%	4.5
net.sourceforge.cobertura.reporting.xml	1	100%	95%	1.524
net.sourceforge.cobertura.util	9	60%	87%	2.992
someotherpackage	1	83%	N/A	1.2

Report generated by Cobertura [illegible]

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

Artifact uploaders

- Tomcat
- JBoss
- Glassfish
- WebSphere
- FTP
- SSH

UI Enhancement

- Dashboard
- Sectioned view
- iPhone/Android

Security Management

- Security Realm
  - LDAP
  - Jenkins's own user database
  - Delegate to servlet container
- Authorization
  - Anyone can do anything
  - Logged-in users can do anything
  - Matrix-based security
  - Project-based Matrix Authorization Strategy
  - Legacy mode

Security Management

- Matrix-based security

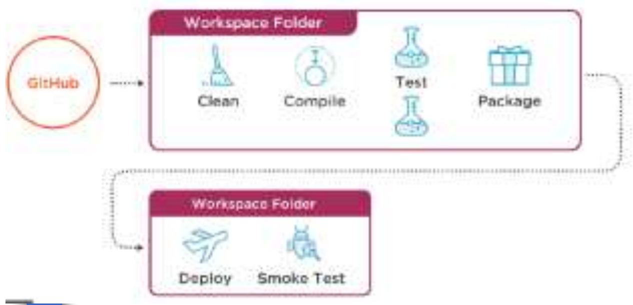
- Project-based Matrix Authorization

Security Management Plugins

- Active directory, OpenID, MySQL, …
- Role based privilege control

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

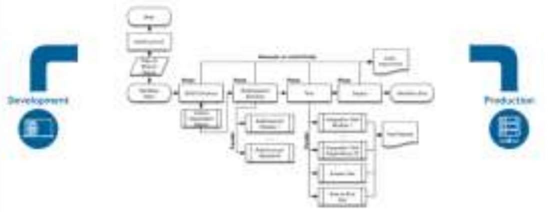
Declarative Vs Scripted Pipeline syntax

- 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

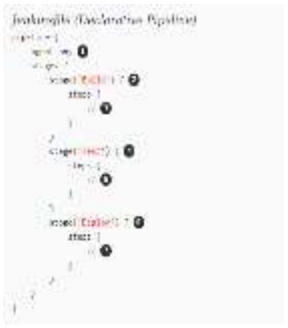


Pipeline Concepts

- 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).

Declarative Pipeline fundamentals

1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

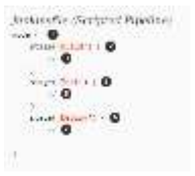


Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any 1
    stages {
        stage('Build') { 2
            steps {
                // 3
            }
        }
        stage('Test') { 4
            steps {
                // 5
            }
        }
        stage('Deploy') { 6
            steps {
                // 7
            }
        }
    }
}
```

Scripted Pipeline fundamentals

1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage. stage blocks are optional in Scripted Pipeline syntax. However, preferred for clearer visualization of each `stage's subset of tasks/steps in the Jenkins UI.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage

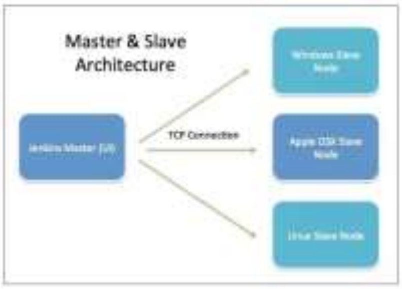


Jenkinsfile (Scripted Pipeline)

```
[illegible]
```

Distributed Environment

Master and Slave Architecture



Jenkins Master

- Your main Jenkins server is the master machine. The tasks performed by the master are :
  - Scheduling build jobs.
  - Dispatching builds to the slaves for the execution.
  - Monitor the slaves.
  - Recording and presenting the build results.
  - Can also execute build jobs directly.

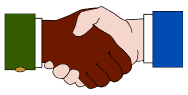
Jenkins Slave

- A slave is a Java executable that runs on a remote machine. The characteristics of the slave are :
  – It hears requests from the Jenkins Master instance.
  – Slaves can run on a variety of operating systems.
  – The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
  – We can configure a project to always run on a particular Slave machine or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com