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

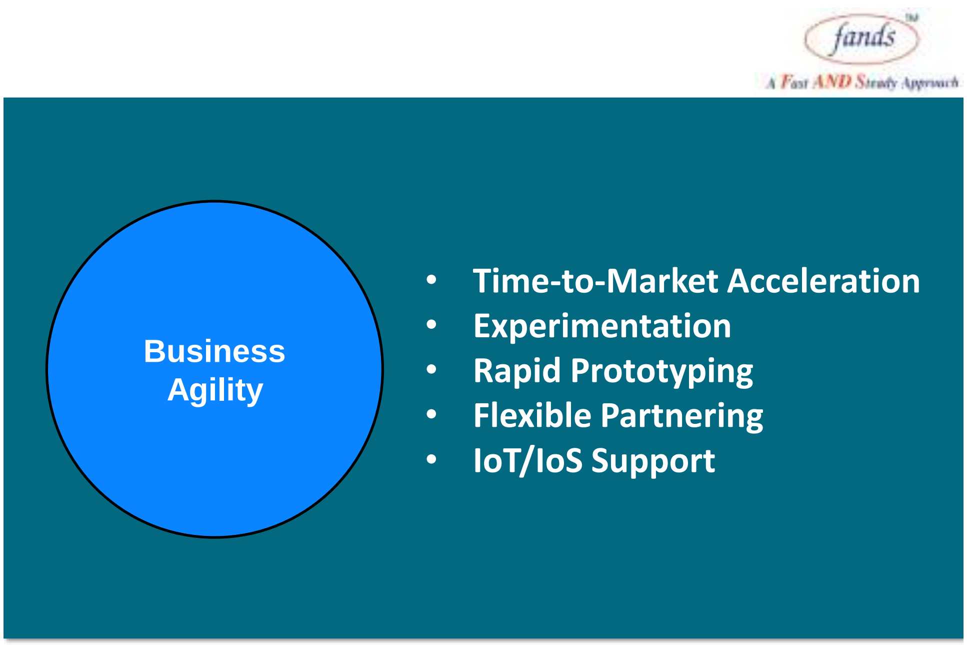
www.fandsindia.com

A Fast AND Steady Approach

Discuss

Why Change?

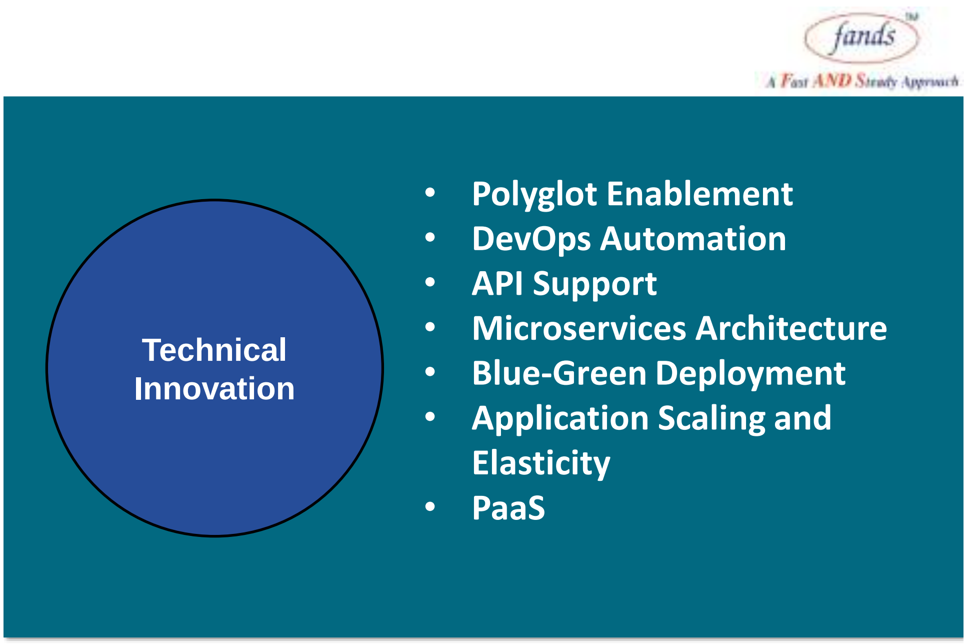
www.fandsindia.com



fands™
A Fast AND Steady Approach

Business Agility

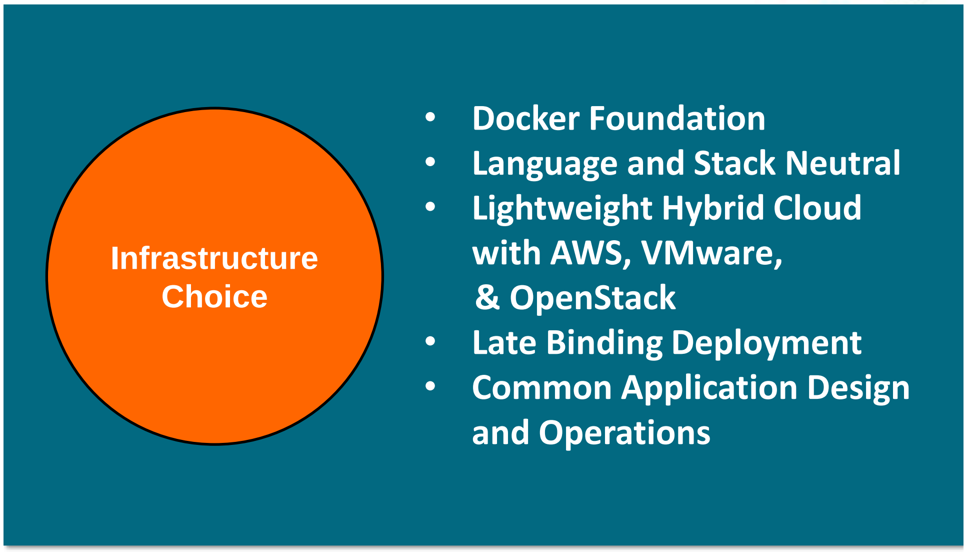
- Time-to-Market Acceleration
- Experimentation
- Rapid Prototyping
- Flexible Partnering
- IoT/IoS Support



fands™
A Fast AND Steady Approach

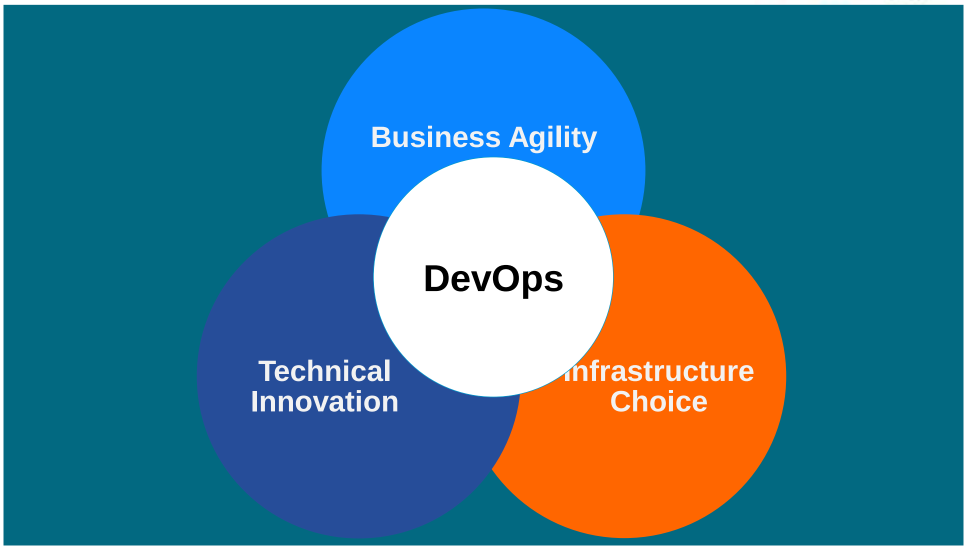
Technical Innovation

- 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



Monoliths to Micro Services

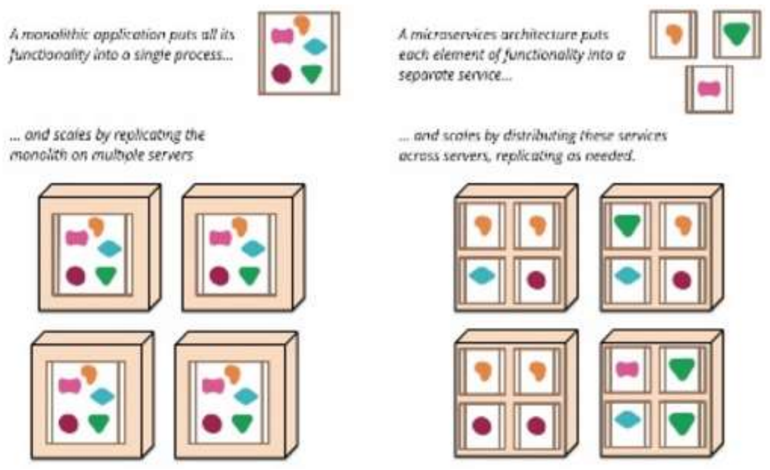
Monoliths to Micro Services

- 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

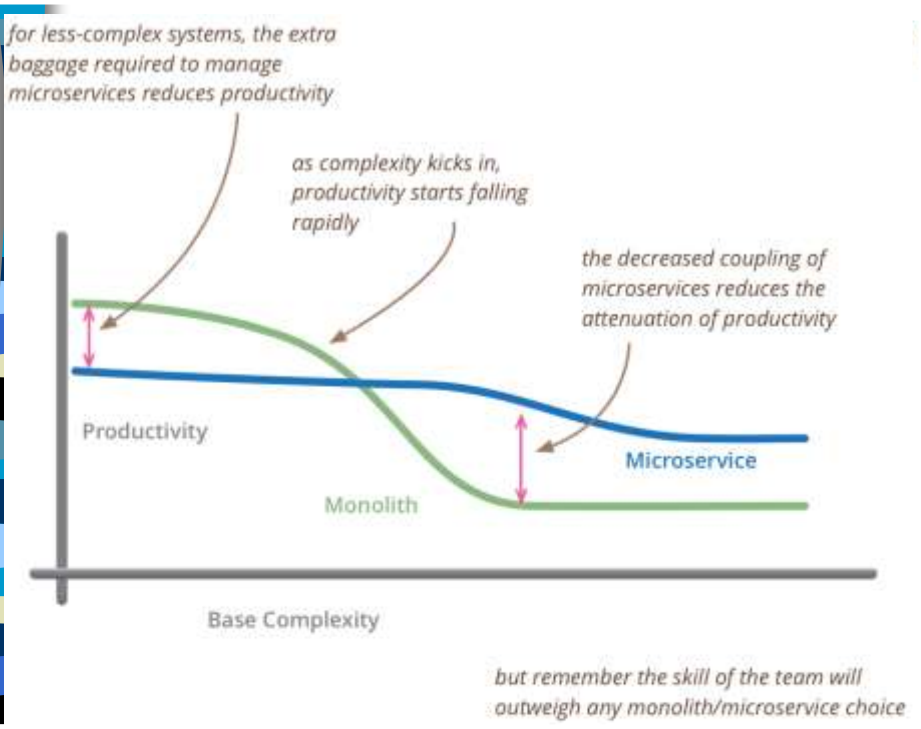

A monolithic application puts all its functionality into a single process...
... and scales by replicating the monolith on multiple servers
A microservices architecture puts each element of functionality into a separate service...
... and scales by distributing these services across servers, replicating 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

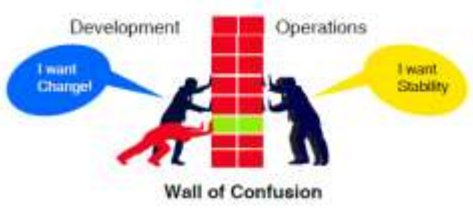
- Benefits of Microservices..?
  - modelled around the business domain
  - deployment automation culture
  - hides implementation details
  - decentralization
  - option to use multiple languages
  - separate deployment
  - separate monitoring
  - isolating problems

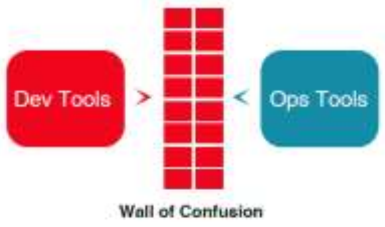


Issues with Micro Services

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

Dev-Ops





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

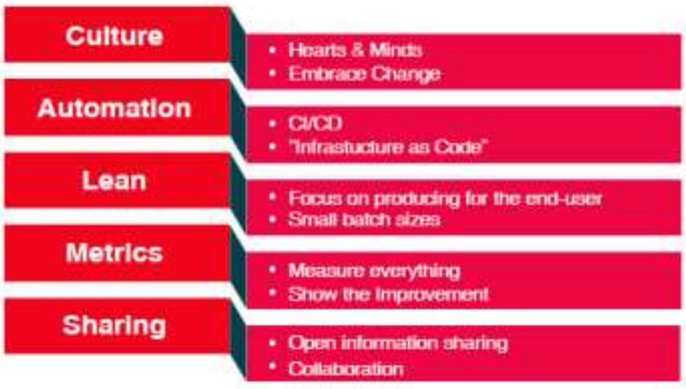
- DevOps is a software development method that highlights collaboration and open communication between teams.
- DevOps teams are composed of developers and operations professionals working together to create sounder, more fail-proof software.
- Teams that have adopted the DevOps ethos have a better handle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

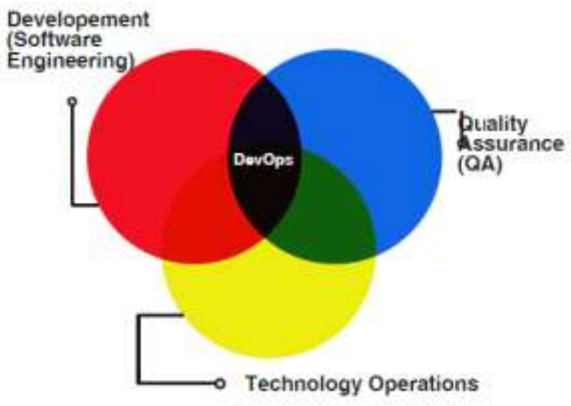
DevOps is a way of thinking.



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



Five Basic Principles of DevOps

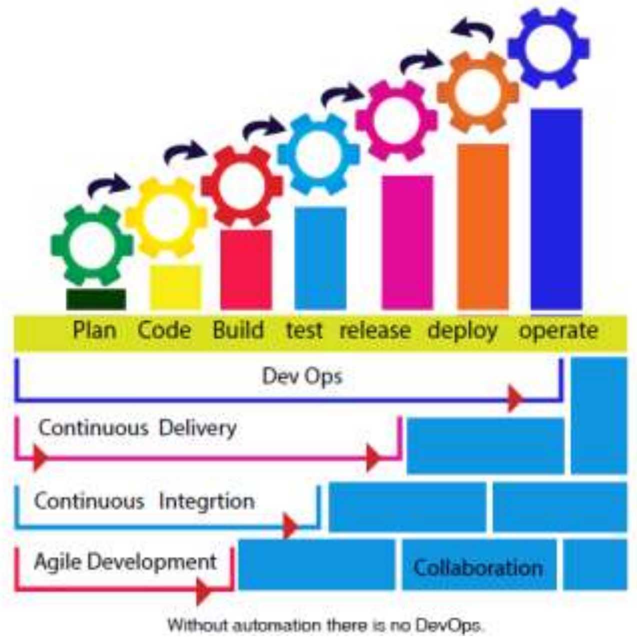
- Eliminate the blame game, Open post-mortems, Feedback, Rewarding failures
- Continous Delivery, Monitoring, Configuration Management
- Business value for end user
- Performance Metrics, Logs, Business goals Metrics,
- People Integration Metrics, KPI
- Ideas, Plans, Goals, Metrics, Complications, Tools

DevOps Combines

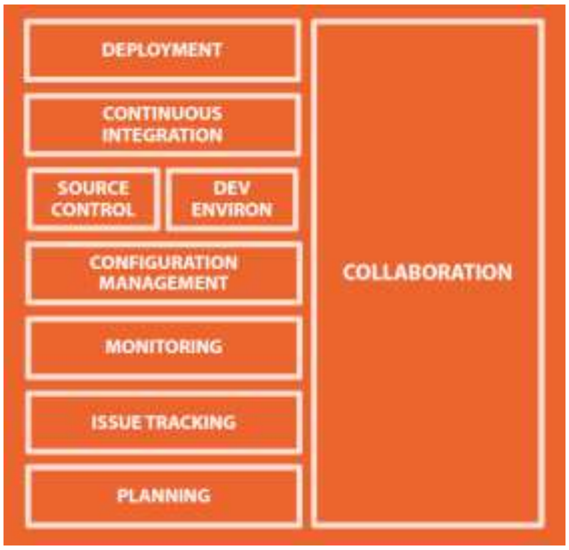
- Develops and verifies against production-like systems
- Reduces cost/time to deliver - Deploy often, deploy faster with repeatable, reliable process
- Increases Quality - Automated testing, Reduce cost/time to test
- Reduces Defect cycle time - Increase the ability to reproduce and fix defects
- Increases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

7Cs OF DevOps

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



DevOps Technology Categories



Collaboration

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

Planning

- Shared Vision
- Tools
  - Trello
  - Visual Studio Online

Issue Tracking

- Rapid Response
- Tools
  - ZENDESK
  - Jira
  - Redmine

Monitoring

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

Configuration Management

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

Source Control

- Controlled Assets
- Tools
  - GitHub
  - …

Development Environment

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

Continuous Integration

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

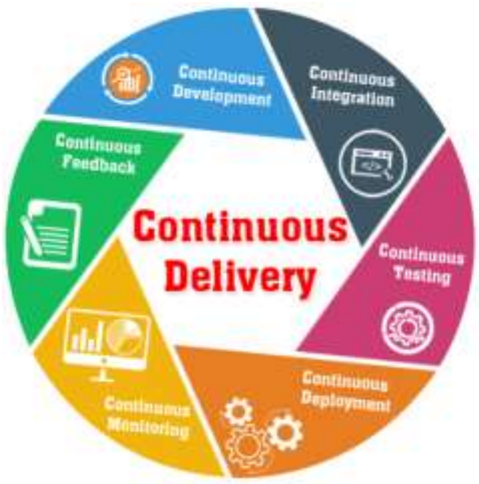
Continuous Deployment

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

fands™

Continuous…

fands™



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.*

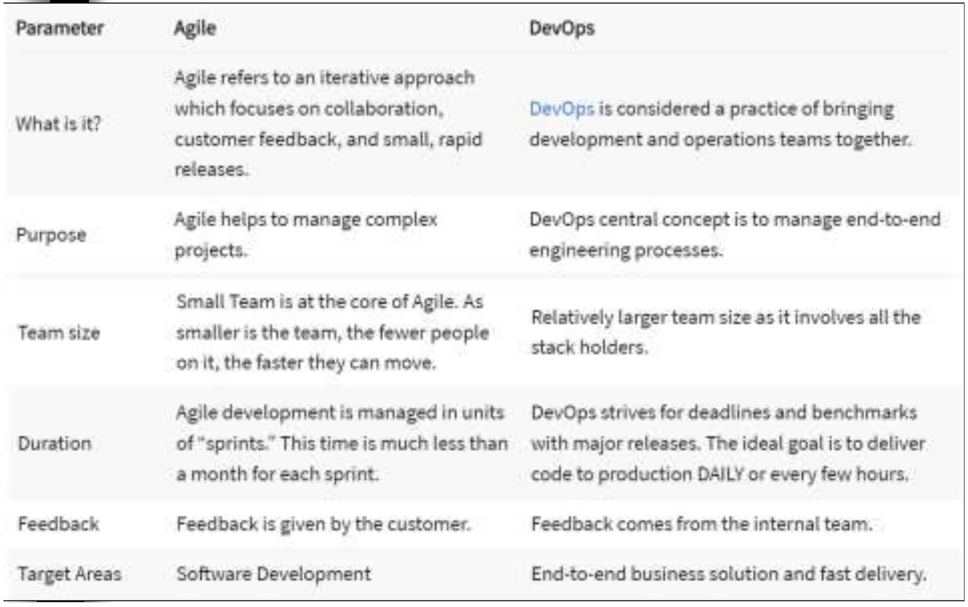
Continuous Deployment

- 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.

Git

Git

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

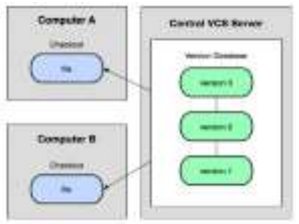
Version Control Systems

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

Why Version Control?

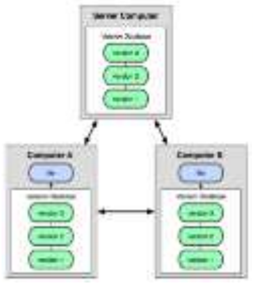
- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes

Centralized VCS



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

Distributed VCS (Git)



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

Why Git?

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

Version Control Terminology

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

Version Control Terminology

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

Version Control Terminology

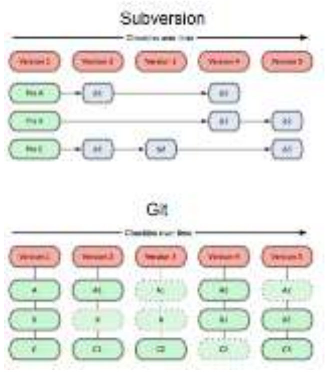
- Commit
  - Git thinks of its data like a set of snapshots of a mini file system.
  - Think of it as a save point during a video game.
- SHA
  - A SHA is basically an ID number for each commit.
  - Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6
- Branch
  - A branch is when a new line of development is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

Version Control Terminology

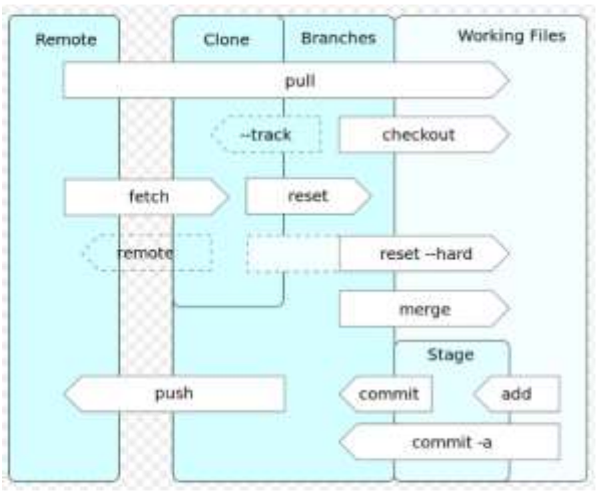
- Working Directory
  - files that you see in your computer's file system. When you open project files up on a code editor, you're working with files in the Working Directory.
- Checkout
  - Content in the repository has been copied to the Working Directory. Possible to checkout many things from a repository;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

Git

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

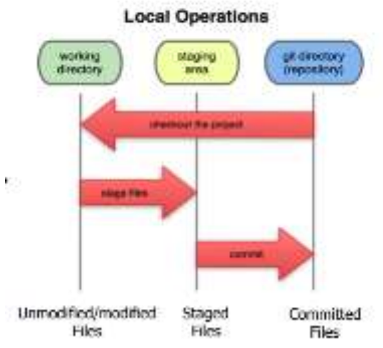


Git

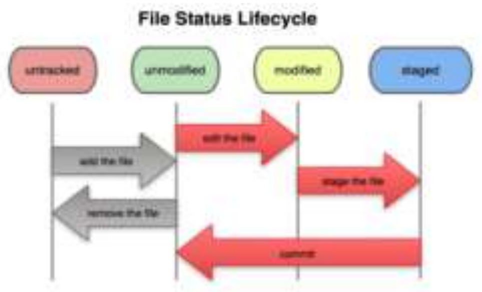


In your local copy on git, files can be:

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



Basic Git Workflow



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

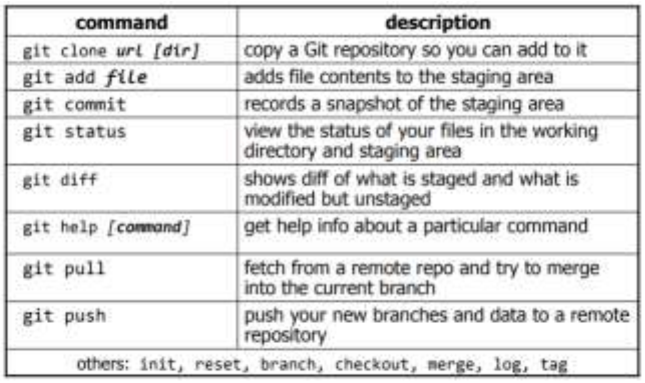
Initial Git configuration

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

Creating a Git Repo

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

Git Commands



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

Add and commit a file

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

Viewing/undoing changes

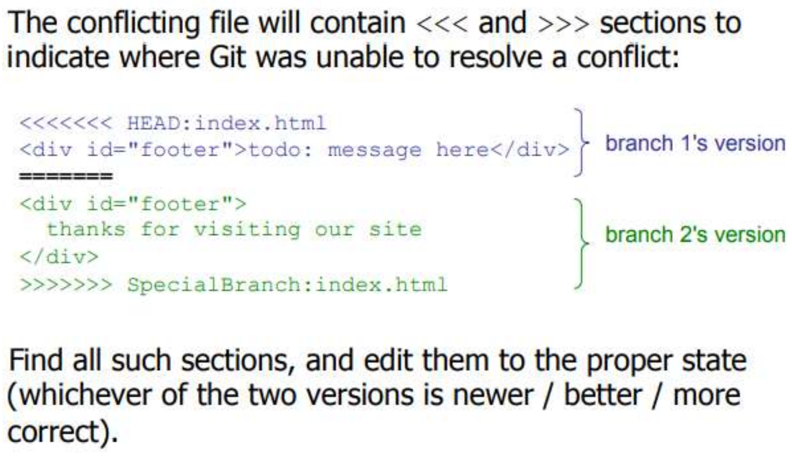
- To view status of files in working directory and staging area:
  - git status or git status –s (short version)
- To see what is modified but unstaged:
  - git diff
- To see a list of staged changes:
  - git diff --cached
- To see a log of all changes in your local repo:
  - git log or git log --oneline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

Branching and Merging

Git uses branching heavily to switch between multiple tasks.

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

Merge Conflicts



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

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

branch 1's version
branch 2's version

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

Interaction with Remote Repo

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

GitHub

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

Demo

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

Jenkins

Continuous Integration (CI)

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

Continuous Integration

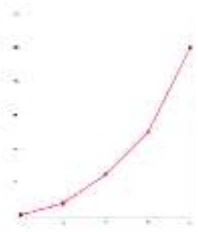
- Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors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

-- Martin Fowler

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

Why Continuous Integration?

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



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

CI – What does it really mean?

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

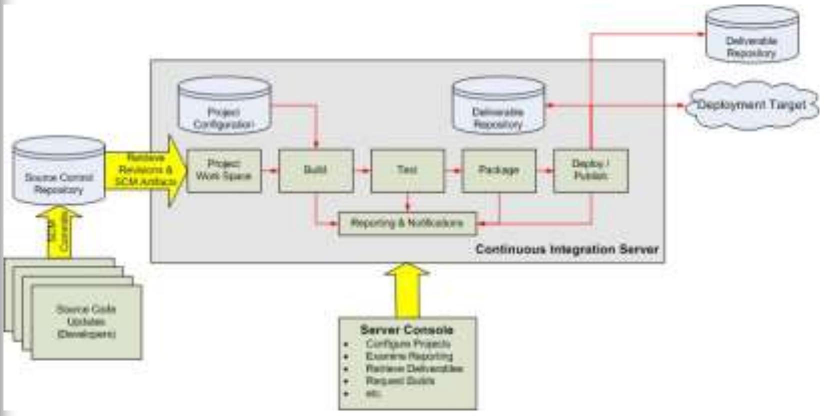
Continuous Integration Benefit

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

Best Practices

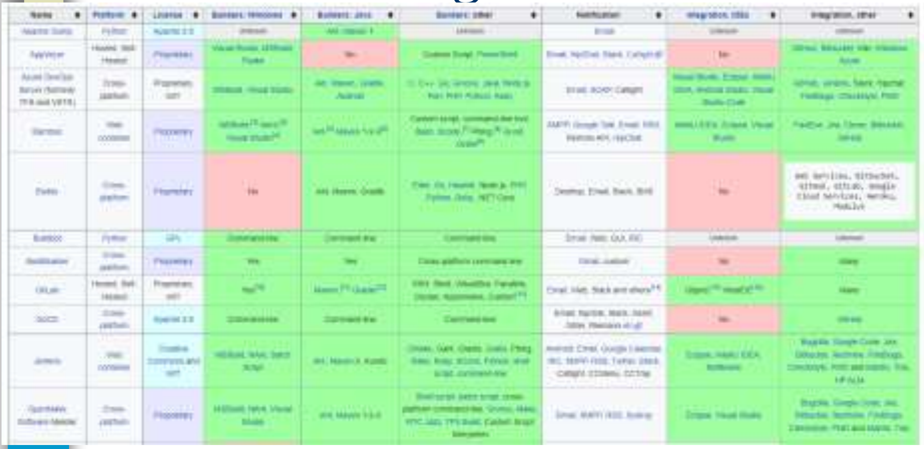
- Single Source Repository
- Automate the Build and Test
- Everyone Commits Every Day
- Keep the Build Fast
- Everyone can see what's happening
- Automate Deployment (Optional)

Continuous Integration Overview



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

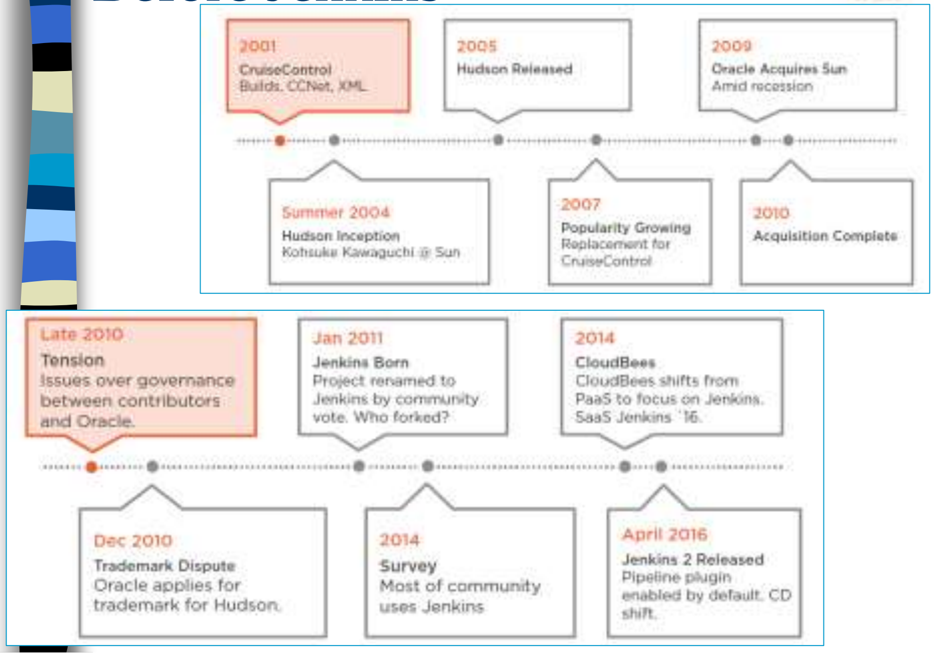
Continuous Integration Tools



Name	Platform	License	Builders: Windows	Builders: Java	Builders: other	Notification	Integration: IDEs	Integration: other
Apache Gump	Python	Apache 2.0	Unknown	[illegible]	Unknown	Email	Unknown	Unknown
AppVeyor	Hosted, Self-Hosted	Proprietary	[illegible]	No	[illegible]	[illegible]	No	[illegible]
Azure DevOps Server [illegible]	Cross-platform	Proprietary, SaaS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Bamboo	Web container	Proprietary	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	Cross-platform	Proprietary	No	[illegible]	[illegible]	[illegible]	No	[illegible]
Buildbot	Python	GPL	Command line	Command line	Command line	[illegible]	Unknown	Unknown
BuildMaster	Cross-platform	Proprietary	Yes	Yes	Cross-platform command line	[illegible]	No	[illegible]
GitLab	Hosted, Self-Hosted	Proprietary, SaaS	Yes	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GoCD	Cross-platform	Apache 2.0	Command line	Command line	Command line	[illegible]	No	[illegible]
Jenkins	Web container	Creative Commons and MIT	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
OpenMake Software Meister	Cross-platform	Proprietary	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

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

Before Jenkins



2001
CruiseControl
Builds, CCNet, XML

Summer 2004
Hudson Inception
Kohsuke Kawaguchi @ Sun

2005
Hudson Released

2007
Popularity Growing
Replacement for CruiseControl

2009
Oracle Acquires Sun
Amid recession

2010
Acquisition Complete

Late 2010
Tension
Issues over governance between contributors and Oracle.

Dec 2010
Trademark Dispute
Oracle applies for trademark for Hudson.

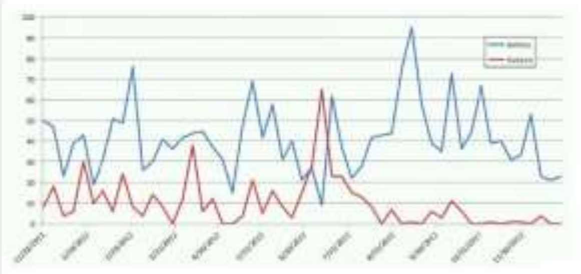
Jan 2011
Jenkins Born
Project renamed to Jenkins by community vote. Who forked?

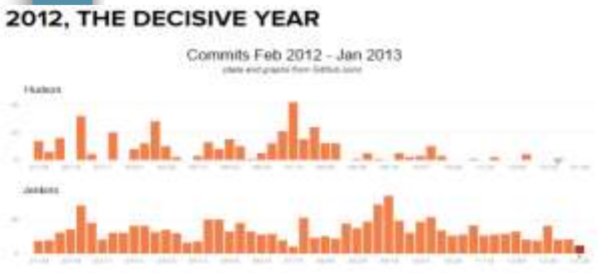
2014
Survey
Most of community uses Jenkins

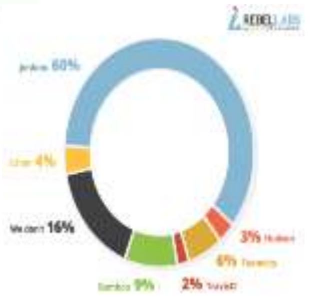
2014
CloudBees
CloudBees shifts from PaaS to focus on Jenkins. SaaS Jenkins '16.

April 2016
Jenkins 2 Released
Pipeline plugin enabled by default. CD shift.

What Happened Next?



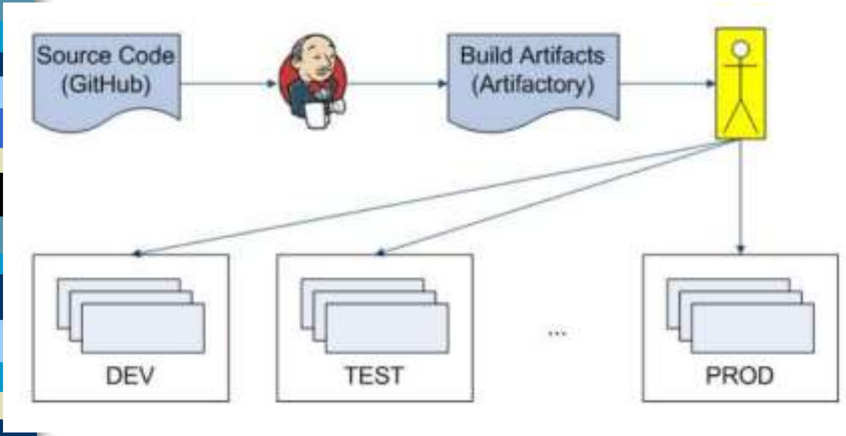




Standard Software Platform

- 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



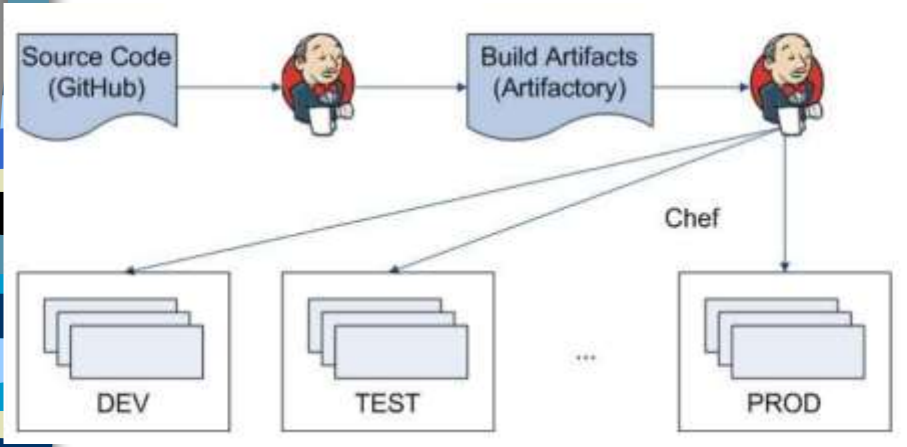
Initial Delivery Pipeline

- 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

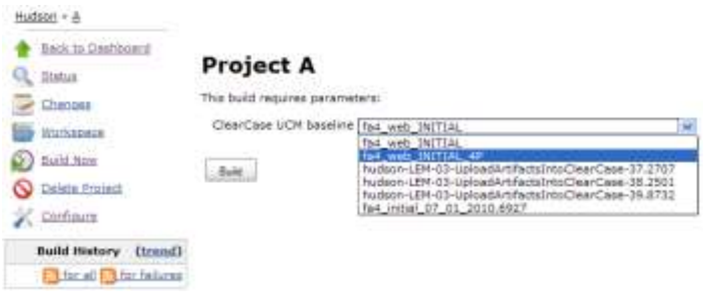
- Manually click build button
- Build periodically
- Build whenever a SNAPSHOT dependency is built
- Build after other projects are built
- Poll SCM
- IRC, Jabber, …

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)



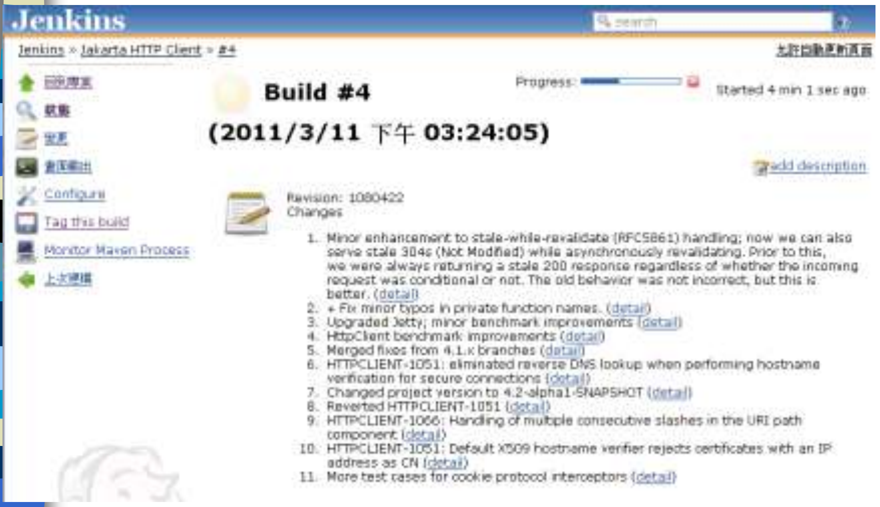
Project A

This build requires parameters:

ClearCase UCM baseline: fs4_web_INITIAL

```
fs4_web_INITIAL
fs4_web_INITIAL_4P
hudson-LEM-03-UploadArtifactsIntoClearCase-37.2707
hudson-LEM-03-UploadArtifactsIntoClearCase-38.2501
hudson-LEM-03-UploadArtifactsIntoClearCase-39.8732
fs4_initial_07_01_2010.6927
```

Code Change History



Jenkins

Jenkins » Jakarta HTTP Client » #4

允許自動更新頁面

- 回到專案
- 狀態
- 變更
- 主控台輸出
- Configure
- Tag this build
- Monitor Maven Process
- 上次建置

Build #4 (2011/3/11 下午 03:24:05)

Progress: Started 4 min 1 sec ago

add description

Revision: 1080422
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 Jetty; 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-1066: Handling of multiple consecutive slashes in the URI path component (detail)
10. HTTPCLIENT-1051: Default X509 hostname verifier rejects certificates with an IP address as CN (detail)
11. More test cases for cookie protocol 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
- …

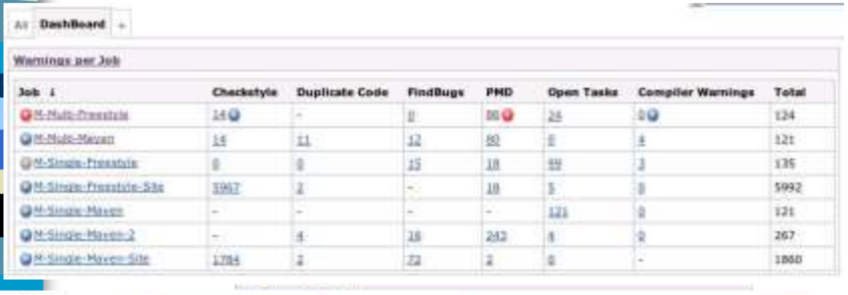
Build Notifier

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

Build Report

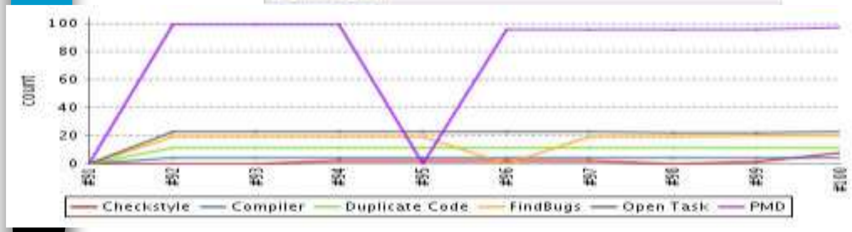
- Static Code Analysis
  - Checkstyle, PMD, Findbugs, Compiler Warning
- Test Report & Code Coverage
  - JUnit, TestNG, Cobertura, Clover
- Open Tasks

Static Code Analysis

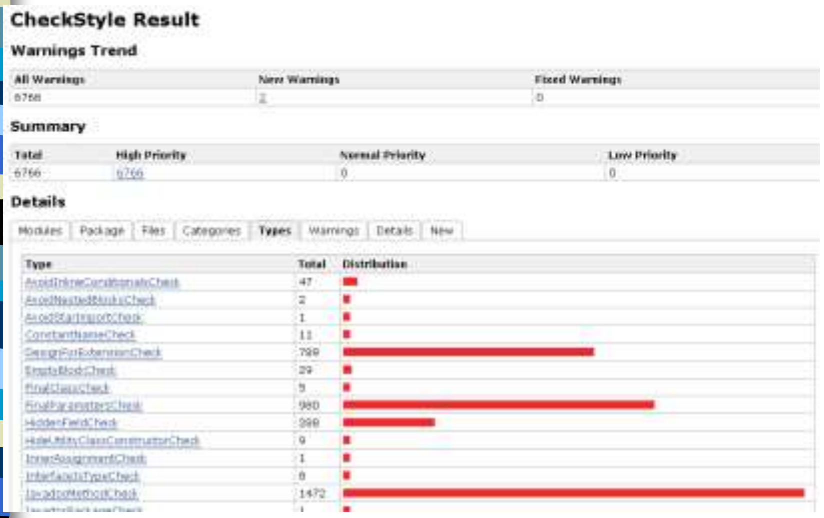

All | DashBoard | +

Warnings per Job

Job ↓	Checkstyle	Duplicate Code	FindBugs	PMD	Open Tasks	Compiler Warnings	Total
M-Multi-Freestyle	14	-	0	86	24	0	124
M-Multi-Maven	14	11	12	80	0	4	121
M-Single-Freestyle	0	0	15	18	99	3	135
M-Single-Freestyle-Site	5967	2	-	18	5	0	5992
M-Single-Maven	-	-	-	-	121	0	121
M-Single-Maven-2	-	4	18	242	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

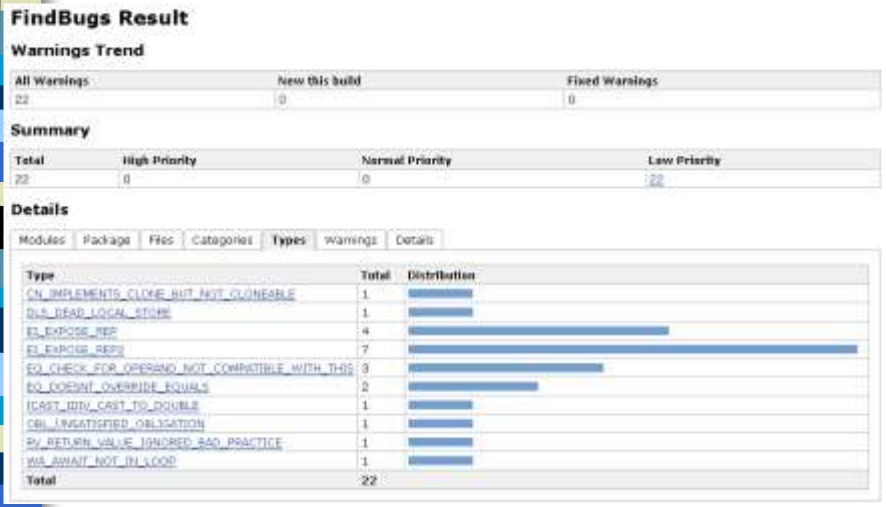
Total	High Priority	Normal Priority	Low Priority
6766	6766	0	0

Details

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

Type	Total	Distribution
AvoidInlineConditionalsCheck	47	
AvoidNestedBlocksCheck	2	
AvoidStarImportCheck	1	
ConstantNameCheck	11	
DesignForExtensionCheck	799	
EmptyBlockCheck	29	
FinalClassCheck	5	
FinalParametersCheck	980	
HiddenFieldCheck	398	
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

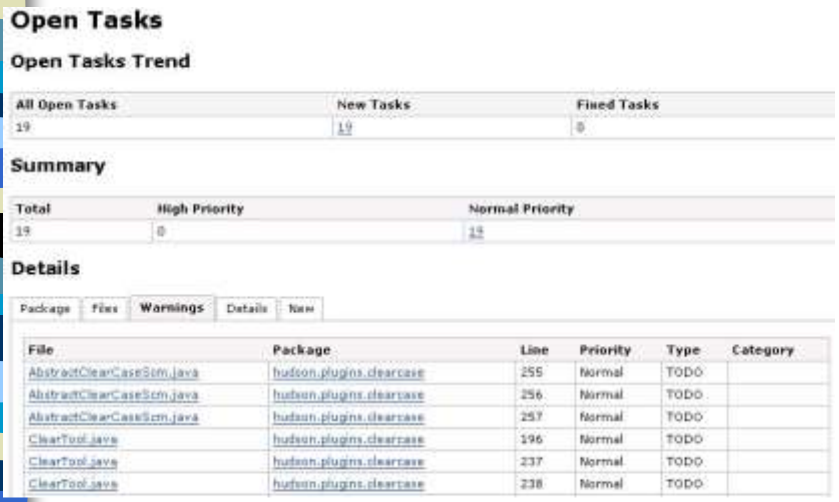
Total	High Priority	Normal Priority	Low Priority
22	0	0	22

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

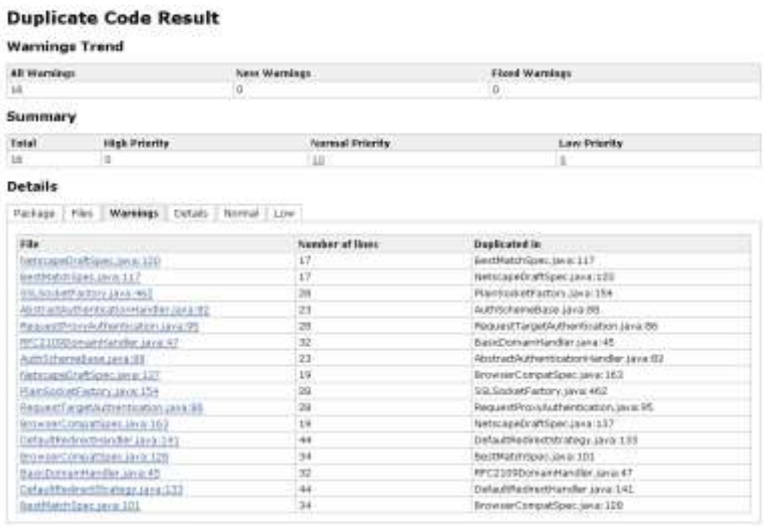
Total	High Priority	Normal Priority
19	0	19

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

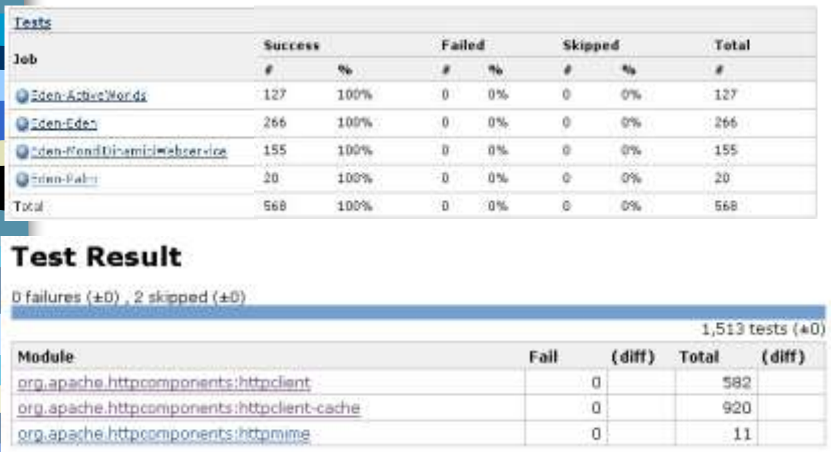
Total	High Priority	Normal Priority	Low Priority
16	0	10	6

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:461	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:162
PlainSocketFactory.java:154	28	SSLSocketFactory.java:462
RequestTargetAuthentication.java:86	28	RequestProxyAuthentication.java:95
BrowserCompatSpec.java:162	19	NetscapeDraftSpec.java:137
DefaultRedirectHandler.java:141	44	DefaultRedirectStrategy.java:133
BrowserCompatSpec.java:120	34	BestMatchSpec.java:101
BasicDomainHandler.java:45	32	RFC2109DomainHandler.java:47
DefaultRedirectStrategy.java:133	44	DefaultRedirectHandler.java:141
BestMatchSpec.java:101	34	BrowserCompatSpec.java:120

Test Report



Tests

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

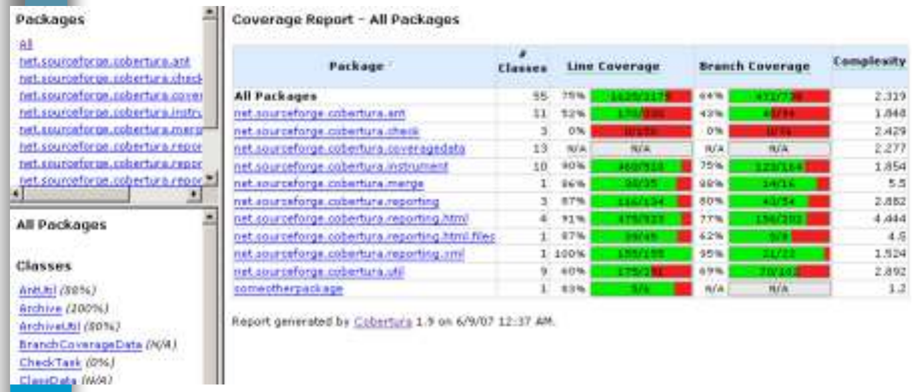
Test Result

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



Packages

All
net.sourceforge.cobertura.ant
net.sourceforge.cobertura.check
net.sourceforge.cobertura.cover
net.sourceforge.cobertura.instru
net.sourceforge.cobertura.merg
net.sourceforge.cobertura.repor
net.sourceforge.cobertura.repor
net.sourceforge.cobertura.repor

All Packages

Classes

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

Coverage Report - All Packages

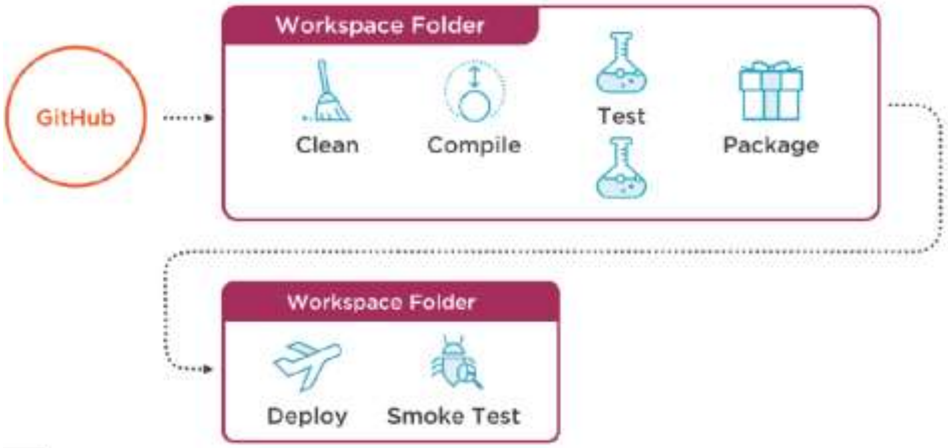
Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	55	75%	1625/2175	64%	471/738	2.319
net.sourceforge.cobertura.ant	11	52%	[illegible]	43%	[illegible]	1.848
net.sourceforge.cobertura.check	3	0%	[illegible]	0%	[illegible]	2.429
net.sourceforge.cobertura.coveragedata	13	N/A	N/A	N/A	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%	[illegible]	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

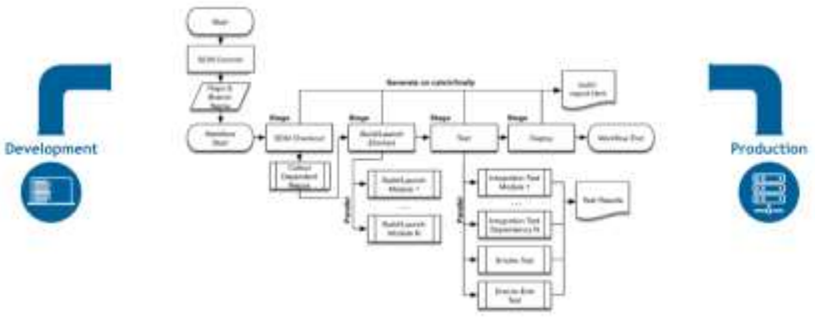
Declarative Vs Scripted Pipeline syntax

- 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.

Virtualization

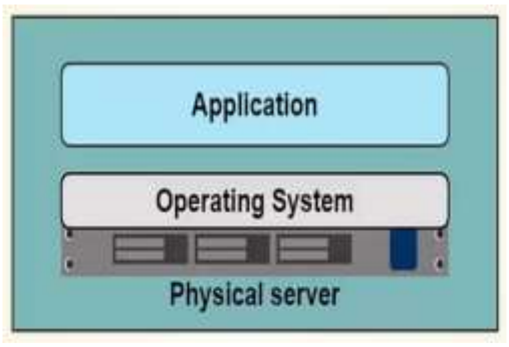
$\downarrow$

Containerization

Virtualization

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

Basic Application Hosting

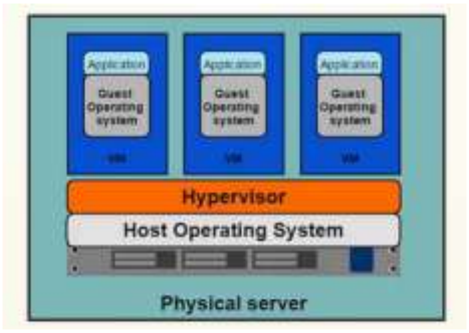


- Problems
  – Slow deployment times
  – Huge costs
  – Wasted resources
  – Difficult to scale or migrate
  – Vendor lock in

www.fandsindia.com

Hypervisor-based Virtualization

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

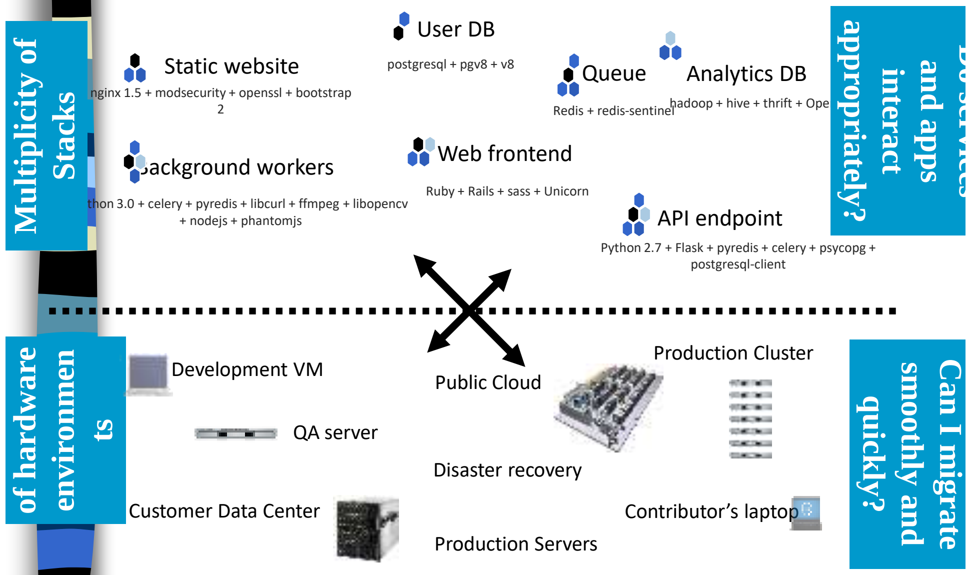


Hypervisor Types

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

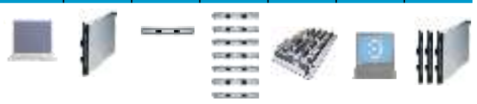
Why Docker?

The Challenge


Multiplicity of Stacks
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + Ope
and apps interact appropriately?
Background workers
thon 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
of hardware environmen ts
Development VM
QA server
Customer Data Center
Public Cloud
Disaster recovery
Production Servers
Production Cluster
Contributor’s laptop
Can I migrate smoothly and quickly?

The Matrix From Hell

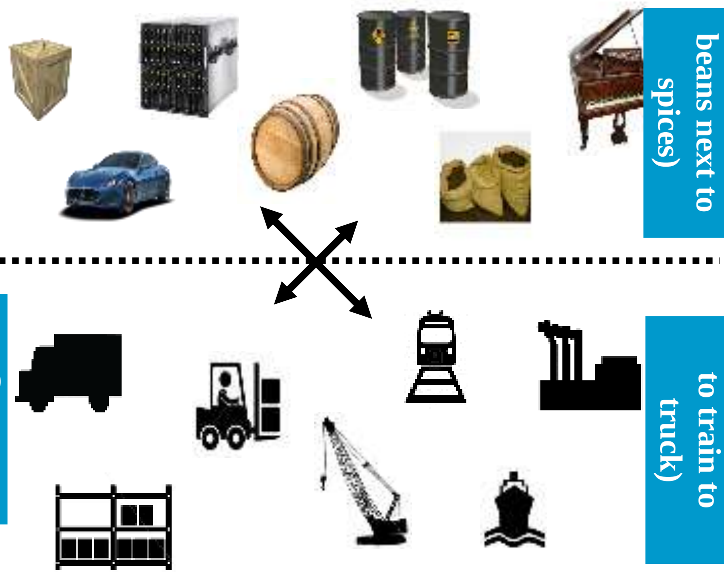
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



Cargo Transport Pre-1960

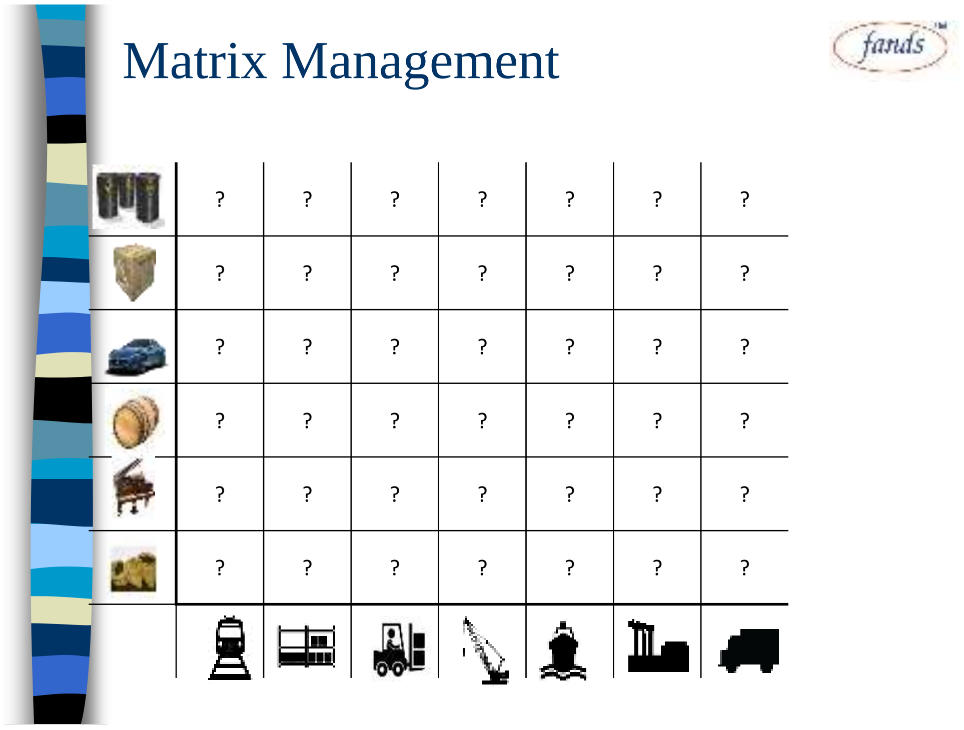
Multiplicity of Goods

methods for transporting/storing



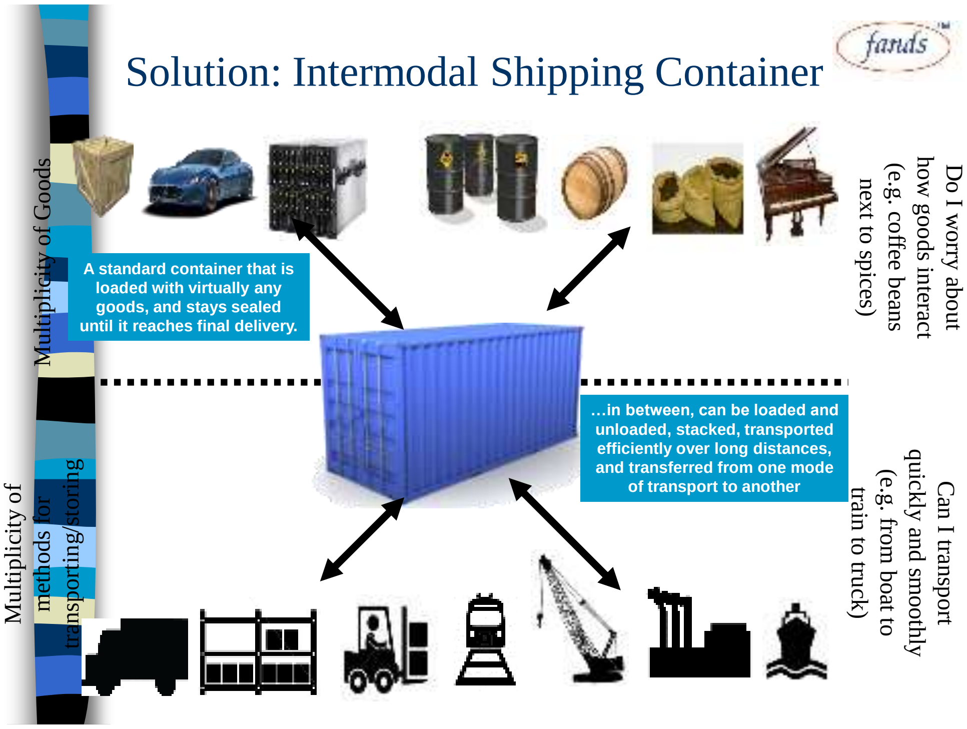
about how goods interact (e.g. coffee beans next to spices)

quickly and smoothly (e.g. from boat to train to truck)



Matrix Management

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?



Solution: Intermodal Shipping Container

Multiplicity of Goods

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

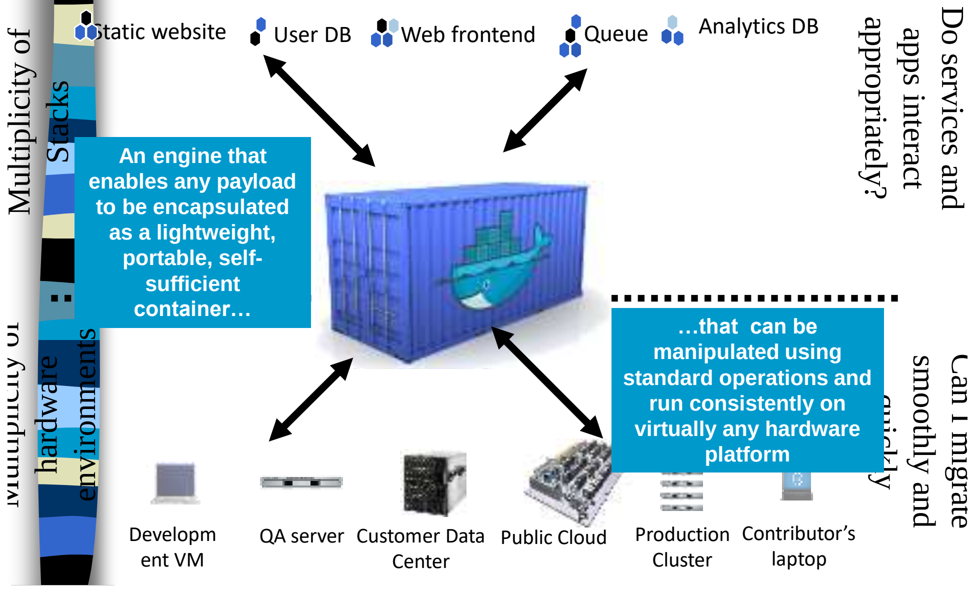
Do I worry about how goods interact (e.g. coffee beans next to spices)

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

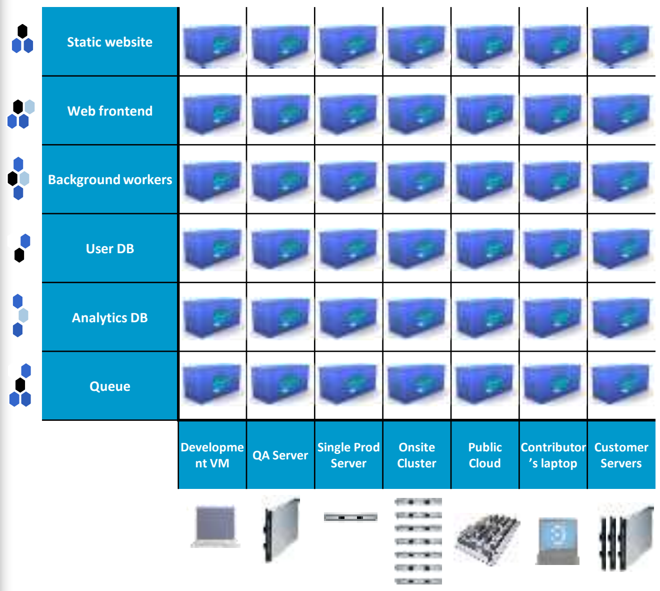
Multiplicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker is a shipping container system for code


Multiplicity of Stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…
…that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiplicity of hardware environments
Can I migrate smoothly and quickly
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop

Docker eliminates the matrix management



	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website							
Web frontend							
Background workers							
User DB							
Analytics DB							
Queue							

Introduction

www.fandsindia.com

What is Docker?

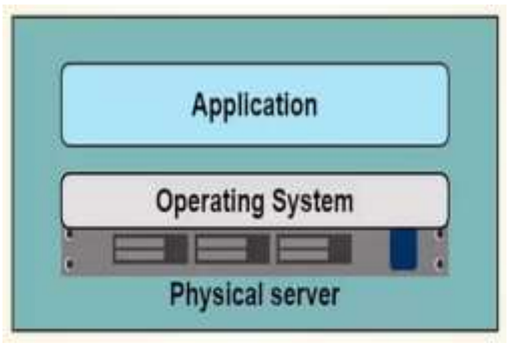
- Docker is a platform for developing, shipping and running applications using container virtualization technology.
- The Docker Platform consists of multiple tools.
  - Docker Engine
  - Docker Hub
  - Docker Machine
  - Docker Swarm
  - Docker Compose
  - Kitematic

www.fandsindia.com

What is Docker?

- Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux, Mac OS and Windows.
  – Wikipedia

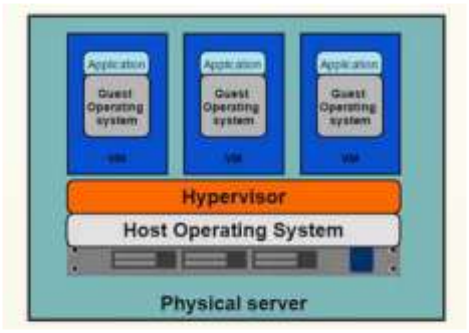
Basic Application Hosting



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

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine



Pros and Cons

- Better resource pooling
  - one physical machine divided into multiple VM
- Easier to scale
- VM's in the cloud
  - Pay as you go

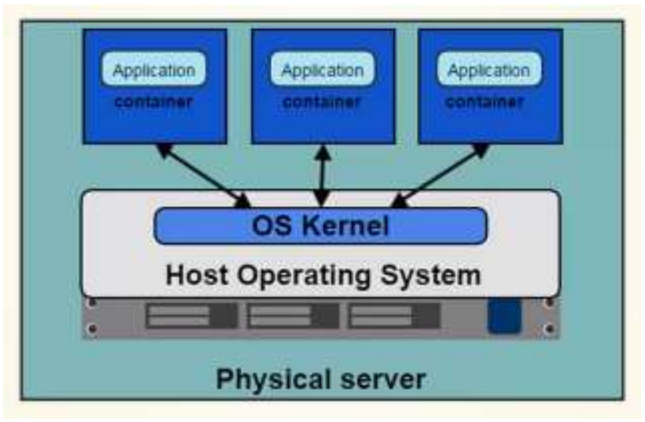
- Each VM stills requires
  - CPU allocation
  - storage
  - RAM
  - An entire guest operation system
- More VM's you run, the more resources you need
- Guest OS means wasted resources

Introducing Containers

Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

- Each guest instance is called a container
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Network ports

Containers

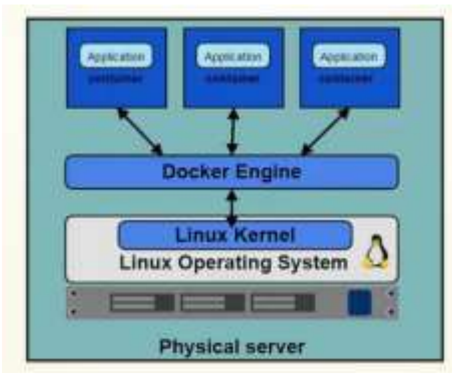


Containers Vs VMs

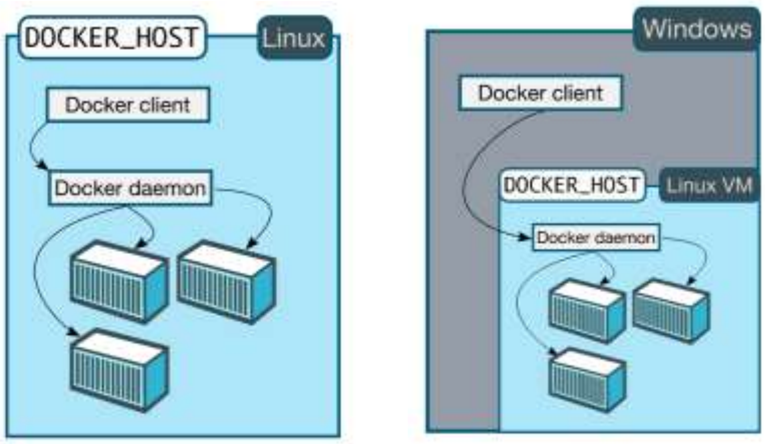
- Containers are more lightweight
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability

Docker Engine

- Docker Engine (deamon) is the program that enables containers to be built, shipped and run.
- It uses Linux Kernel namespaces and control groups
- Namespaces give us the isolated workspace



Docker Engine on Different OS

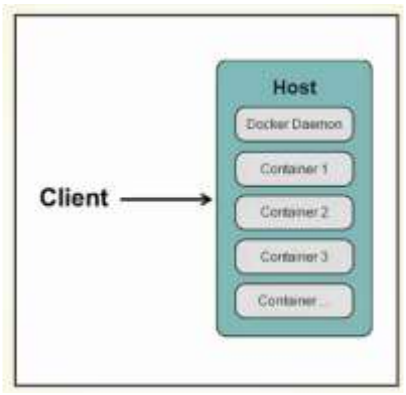


Installation

- WINDOWS AND MAC OS X:
  - Docker need linux kernel... so you will need a linux Virtual Machine. Docker toolbox contains everything needed.
- VERIFY DOCKER INSTALLATION
  - $ docker run hello-world
    - This command will download(if required) and run an hello world container

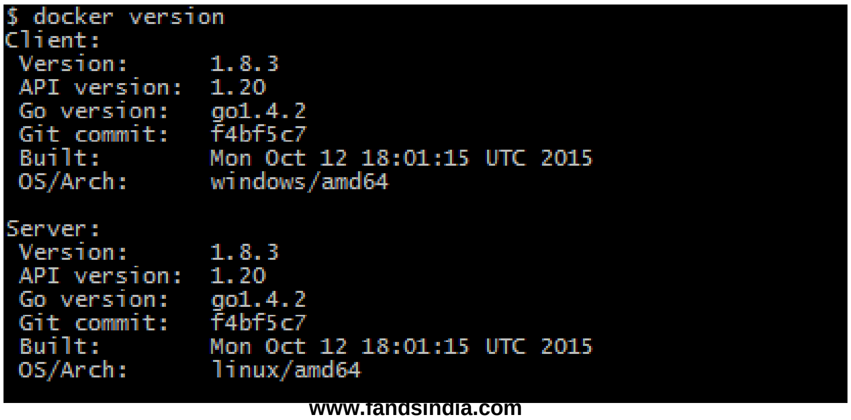
Docker Client and Daemon

- Client / Server Architecture
- Client takes user inputs and send them to the daemon
- Daemon builds, runs, and distributes containers
- Client and daemon can run on same or different hosts



Checking Client and Daemon Version

- $ docker version



```
$ docker version
Client:
 Version:      1.8.3
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   f4bf5c7
 Built:        Mon Oct 12 18:01:15 UTC 2015
 OS/Arch:      windows/amd64

Server:
 Version:      1.8.3
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   f4bf5c7
 Built:        Mon Oct 12 18:01:15 UTC 2015
 OS/Arch:      linux/amd64
```

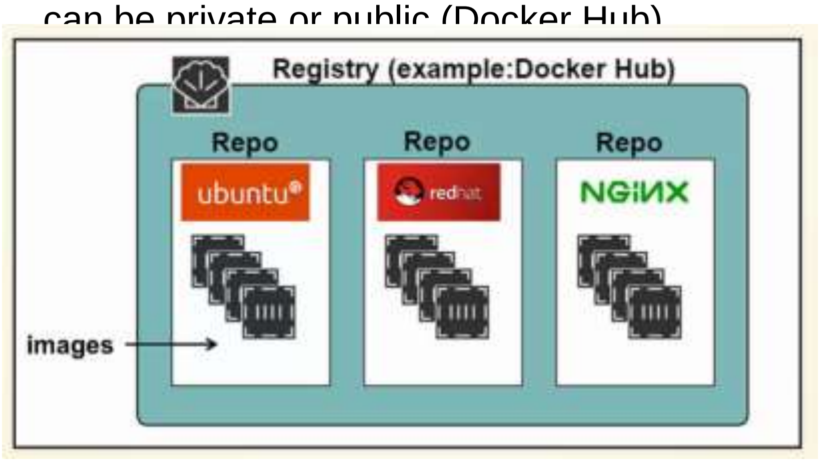
Images and Containers

- Images
  - Read only template used to create containers
  - Built by you or other Docker users
  - Stored in the Docker Hub or your local Registry
- Containers
  - Isolated application platform
  - Contains everything needed to run your application
  - Based on images

www.fandsindia.com

Registry & Repository

- Registry is where we store images. Registry can be private or public (Docker Hub)



Docker Orchestration

- Three tools for orchestrating distributed applications with Docker
  - Docker Machine
    - Tool that provisions Docker hosts and installs the Docker Engine on them
  - Docker Swarm
    - Tool that clusters many Engines and schedules containers
  - Docker Compose
    - Tool to create and manage multi-container applications

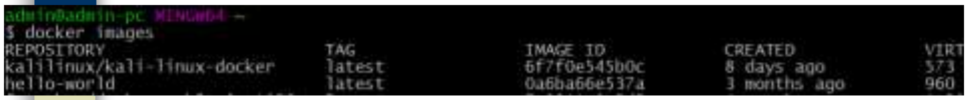
Benefits of Docker

- Separation of concerns
  - Developers focus on building their apps
  - System administrators focus on deployment
- Fast development cycle
- Application portability
  - Build in one environment, ship to another
- Scalability
  - Easily spin up new containers if needed
- Run more apps on one host machine

Images

Display Local Images

- When creating a container, docker will attempt to use a local image first
- If no local image is found, the Docker daemon will look in Docker Hub unless another registry is specified.



```
admin@admin-pc MINGW64 ~
$ docker images
REPOSITORY                     TAG           IMAGE ID        CREATED          VIRT
kalilinux/kali-linux-docker    latest        6f7f0e545b0c    8 days ago       573
hello-world                    latest        0a6ba66e537a    3 months ago     960
```

Image Tags

- Images are specified by repository:tag
- The same image may have multiple tags
- Default tag is latest
- Classically Tags are version or tools
- Lookup the repository on Docker Hub to see what tags are available

Getting Started With Containers

Creating a container

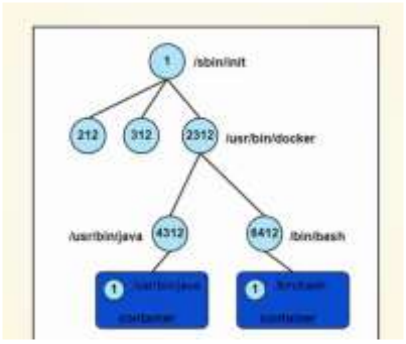
- Using docker run
  - Syntax:
  - $ docker run [options] [image] [command] [args]
    - image is specified with repository:tag
- examples:
  - docker run ubuntu:14.04 echo “HelloWorld”

Container With Terminal

- Use some options:
  - -i flag tells docker to connect to STDIN on the container
  - -t flag specifies to get a pseudo-terminal
- EXAMPLE
  - docker run -i -t ubuntu /bin/bash

Container Processes

- A container only runs as long as the process from your command is running
- Your command's process is always PID 1 inside the container



Container ID

- Containers can be specified using their ID or name
- Long ID and short ID
- Short ID and name can be obtained using docker ps command to list containers
- Long ID obtained by inspecting a container

Find Your Containers

- use docker ps to list running containers
- use docker ps -a to list all containers (includes containers that are stopped)

Running In Detached Mode

- Also known as running in the background or as a daemon
- Use -d flag
- To observe output, use docker logs [container id]
  - docker run -d ubuntu ping 127.0.0.1 -c 50

Ports Mapping

- Run a web application inside a container
- The -P flag to map container ports to host ports

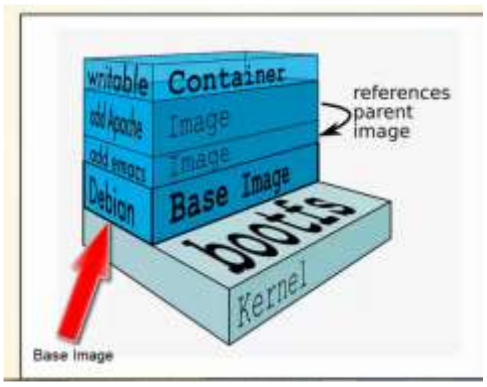
Practical Container

- Run a web app inside a container
- The –p flag to map container ports to host ports

Building Images

Image Layers

- Images are comprised of multiple layers
- A layer is also just another image
- Every image contains a base layer
- Docker uses a copy on write system
- Layers are read only



The Container Writable Layer

- Docker creates a top writable layer for containers
- Parent images are read only
- All changes are made at the writeable layer

Docker Commit

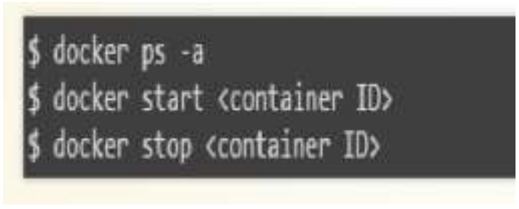
- **Docker commit** command saves changes in a container as a new image
  - docker commit [options] [container ID] [repository:tag]
  - Repository name should be based on username/application
- Can reference the container with container name instead of ID
  - docker commit <id> lodelestra/vim:1.0

Managing Images And Containers

www.fandsindia.com

Start And Stop Containers

- Find your containers first with docker ps and note the ID or name
- 'docker start' and 'docker stop'



```
$ docker ps -a
$ docker start <container ID>
$ docker stop <container ID>
```

www.fandsindia.com

Getting Terminal Access

- Use docker exec command to **start another process** within a container
- Execute /bin/bash to get a bash shell
- docker exec -i -t [container ID] /bin/bash
- Exiting from the terminal
  will **not** terminate the container

Deleting Containers

- Can only delete containers that have been stopped
- Use docker rm command
- Specify the container ID or name

Deleting Local Images

- Use docker rmi command
- Syntax
  docker rmi [image ID]
  or
  docker rmi [repo:tag]
- if an image is tagged multiple times, you have to remove each tag

Renaming Tagging Images

- Used to rename a local image repository before pushing to Docker Hub
- Syntax
  docker tag [image ID] [repo:tag]
  or
  docker tag [local repo:tag] [Docker Hub repo:tag]

Docker Hub Repositories

- Users can create their own repositories on Docker Hub
- Public and Private (one free)
- Push local images to a repository

Pushing Images To Docker Hub

- Login to your Docker hub account
- Create a new public repository
- Tag your local image
- Push new image
- Confirm in docker repository
- Use docker push command
- Syntax
  docker push [repo:tag]
- Local repo must have same name and tag as the Docker Hub repo

Volumes

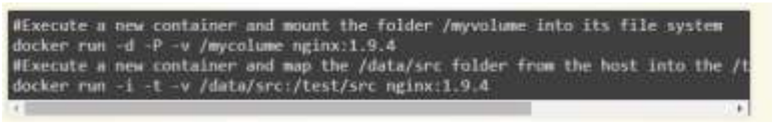
Volumes

A volumes is a designated directory in a container, which is designed to persist data, independent of the container's file cycle

- Volume changes are excluded when updating an image
- Persist when a container is deleted
- Can be mapped to a host folder
- Can be shared between containers

Mount A Volume

- Volumes are mounted when creating or executing a container
- Can be mapped to a host directory
- Volume paths specified must be absolute

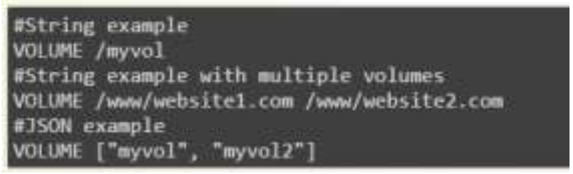


```
#Execute a new container and mount the folder /myvolume into its file system
docker run -d -P -v /mycolume nginx:1.9.4
#Execute a new container and map the /data/src folder from the host into the /t
docker run -i -t -v /data/src:/test/src nginx:1.9.4
```

www.fandsindia.com

Volumes In Dockerfile

- VOLUME instruction creates a mount point
- Can specify arguments JSON array or string
- Cannot map volumes to host directories
- Volumes are initialized when the container is executed



```
#String example
VOLUME /myvol
#String example with multiple volumes
VOLUME /www/website1.com /www/website2.com
#JSON example
VOLUME ["myvol", "myvol2"]
```

www.fandsindia.com

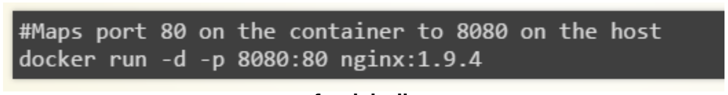
Uses Of Volumes

- De-couple the data that is stored from the container which created the data (example logs)
- Good for sharing data between containers
  - Can setup a data containers which has a volume you mount in other containers
- Mounting folders from the host is good for testing purpose but generally not recommended for production use

Container Networking Basics

Mapping Ports

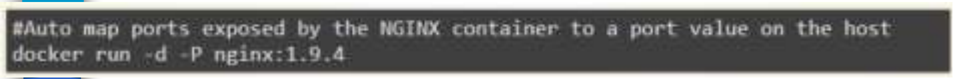
- **Recall:** containers have their own network and IP address
- Map exposed container ports to ports on the host machine
- Ports can be manually mapped or auto mapped
- Uses the -p and -P parameters in docker run



```
#Maps port 80 on the container to 8080 on the host
docker run -d -p 8080:80 nginx:1.9.4
```

Automapping Ports

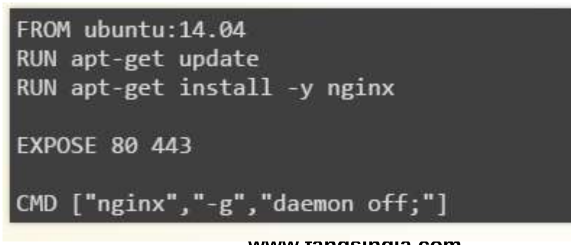
- Use the -P option in **docker run**
- Automatically maps exposed ports in the container to a port number in the host
- Host port numbers used go from 49153 to 65535
- Only work for ports defined in the EXPOSE instruction



```
#Auto map ports exposed by the NGINX container to a port value on the host
docker run -d -P nginx:1.9.4
```

Expose Instruction

- Configures which ports a container will listen on at runtime
- Ports still need to be mapped when container is executed



```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y nginx

EXPOSE 80 443

CMD ["nginx","-g","daemon off;"]
```

www.fandsindia.com

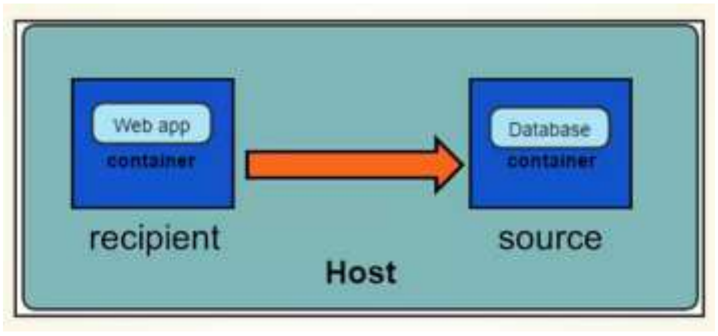
Linking Containers

***Linking** is a communication method between containers which allows them to securely transfer data from one to another*

- Source and recipient containers
- Recipient containers have access to data on source containers
- Links are established based on container names

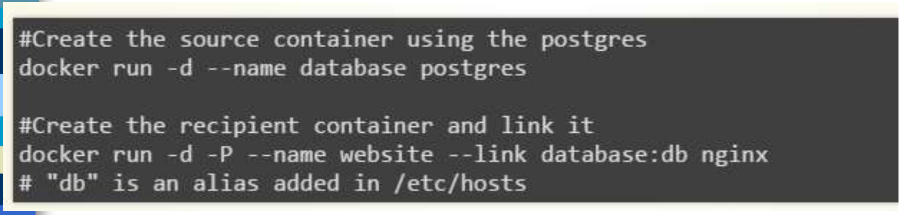
www.fandsindia.com

Linking Containers



Creating A Link

- Create the source container first (database)
- Create the recipient container and use the --link option



```
#Create the source container using the postgres
docker run -d --name database postgres

#Create the recipient container and link it
docker run -d -P --name website --link database:db nginx
# "db" is an alias added in /etc/hosts
```

Uses Of Linking

- Containers can talk to each other without having to expose ports to the host
- Essential for micro service application architecture
- Example:
  – Container with Tomcat running
  – Container with MySQL running
  – Application on Tomcat needs to connect to MySQL

Deploying a registry server

Private Registry

- Running on localhost
  - docker **run** -d -p 5000:5000 --restart=always --name registry registry:2
- Tag image to local registry
  - docker tag hello-world localhost:5000/hello-world
- Push to your registry
  - docker push localhost:5000/hello-world

Private Registry

- Pull it back from your registry:
  - docker pull localhost:5000
- To stop your registry, you would:
  - docker stop registry && docker rm -v registry

Configure Storage

- By default, your registry data is persisted as a Docker volume on the host file system
  - docker run -d -p 5000:5000 --restart=always --name registry -v \`pwd\`/data:/var/lib/registry registry:2

Running a domain registry

- While running on localhost has its uses, most people want their registry to be more widely available.
  - To do so, the Docker engine requires you to secure it using TLS, which is conceptually very similar to configuring your web server with SSL.
  - Get a certificate
    - Configure with -e

Restricting access

- Native basic auth
  - The simplest way to achieve access restriction is through basic authentication
    - First create a password file with one entry for sample user user & password
      - mkdir auth
      - docker **run** --**entrypoint** htpasswd registry:2 -Bbn testuser testpassword > auth/htpasswd

Dockerfile

Need of Dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

Usage

- Docker build command builds an image from a Dockerfile and a context. The build's context is the files at a specified location PATH or URL. The PATH is a directory on your local filesystem. The URL is a the location of a Git repository.
- A context is processed recursively. So, a PATH includes any subdirectories and the URL includes the repository and its submodules. A simple build command that uses the current directory as context:

Usage

- $ docker build .
  - Sending build context to Docker daemon 6.51 MB ...
- The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it's best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the files needed for building the Dockerfile.

Implementation

- Traditionally, the Dockerfile is called Dockerfile and located in the root of the context. You use the -f flag with docker build to point to a Dockerfile anywhere in your file system
  - docker build -f /path/to/a/Dockerfile .
  - docker build -t shykes/myapp .
  - docker build -t shykes/myapp:1.0.2 -t shykes/myapp:latest .
- The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image. The Docker daemon will automatically clean up the context you sent.

Format

- Here is the format of the Dockerfile:
  - *# Comment*
    - INSTRUCTION arguments The instruction is not case-sensitive, however convention is for them to be UPPERCASE in order to distinguish them from arguments more easily.
- Docker runs the instructions in a Dockerfile in order.
  - **The first instruction must be `FROM`** in order to specify the *Base Image*

Comment

- Docker will treat lines that begin with # as a comment.
- A # marker anywhere else in the line will be treated as an argument.
- This allows statements like:
  - # Comment
  - RUN echo 'we are running some # of cool things'

Environment Replacement

- Environment variables (declared with the ENV statement) can also be used in certain instructions as variables to be interpreted by the Dockerfile. Escapes are also handled for including variable-like syntax into a statement literally.
- Environment variables are notated in the Dockerfile either with $variable_name or ${variable_name}. They are treated equivalently and the brace syntax is typically used to address issues with variable names with no whitespace, like ${foo}_bar

Environment Replacement

- The ${variable_name} syntax also supports a few of the standard bash modifiers as specified below:
  - ${variable:-word} indicates that if variable is set then the result will be that value. If variable is not set then word will be the result.
  - ${variable:+word} indicates that if variable is set then word will be the result, otherwise the result is the empty string.
- In all cases, word can be any string, including additional environment variables.
- Escaping is possible by adding a \ before the variable: \$foo or \${foo}, for example, will translate to $foo and ${foo} literals respectively.

ENV

```
FROM busybox
ENV foo /bar
WORKDIR ${foo}
# WORKDIR /bar
ADD . $foo
# ADD . /bar
COPY \$foo /quux
# COPY $foo /quux
```

- Supported by
  - ADD
  - COPY
  - ENV
  - EXPOSE
  - LABEL
  - USER
  - WORKDIR
  - VOLUME
  - STOPSIGNAL

Command Processing Basic

- Environment variable substitution will use the same value for each variable throughout the entire command.

```
ENV abc=hello
ENV abc=bye def=$abc
ENV ghi=$abc
```

  - will result in def having a value of hello, not bye. However, ghi will have a value of bye because it is not part of the same command that set abc to bye.

.dockerignore file

- Before the docker CLI sends the context to the docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.
  /temp
  //temp*
  temp?

Exceptions

- Lines starting with ! (exclamation mark) can be used to make exceptions to exclusions. The following is an example .dockerignore file that uses this mechanism:
  *.md
  !**README**.md
  - All markdown files *except* README.md are excluded from the context.

FROM

FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>

- The FROM instruction sets the Base Image for subsequent instructions. As such, a valid Dockerfile must have FROM as its first instruction. The image can be any valid image

MAINTAINER

- MAINTAINER <name>
- The MAINTAINER instruction allows you to set the Author field of the generated images.

RUN

- RUN has 2 forms:
  - RUN <command> (*shell* form, the command is run in a shell - /bin/sh -c)
  - RUN ["executable", "param1", "param2"] (*exec* form)
- The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.

RUN

- Layering RUN instructions and generating commits conforms to the core concepts of Docker where commits are cheap and containers can be created from any point in an image's history, much like source control.
- The exec form makes it possible to avoid shell string munging, and to RUN commands using a base image that does not contain /bin/sh.
- In the shell form you can use a \ (backslash) to continue a single RUN instruction onto the next line. For example, consider these two lines:

Continuation of RUN

- RUN /bin/bash -c 'source $HOME/.bashrc ;\
- echo $HOME'
- Together they are equivalent to this single line:

- RUN /bin/bash -c 'source $HOME/.bashrc ; echo $HOME'

CMD

- The CMD instruction has three forms:
  - CMD ["executable","param1","param2"] (exec form, this is the preferred form)
  - CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
  - CMD command param1 param2 (shell form)
- There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.

CMD

- **The main purpose of a CMD is to provide defaults for an executing container.** These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.

Note:

- Don’t confuse RUN with CMD.
  RUN actually runs a command and commits the result; CMD does not execute anything at build time, but specifies the intended command for the image

LABEL

- LABEL <key>=<value> <key>=<value> <key>=<value> ...
- The LABEL instruction adds metadata to an image. A LABEL is a key-value pair. To include spaces within a LABEL value, use quotes and backslashes as you would in command-line parsing.

LABEL

- **LABEL** "com.example.vendor"="ACME Incorporated"
- **LABEL** com.example.label-with-value="foo"
- **LABEL** version="1.0"

- An image can have more than one label. To specify multiple labels, Docker recommends combining labels into a single LABEL instruction where

EXPOSE

- **EXPOSE** <port> [<port>...]
- The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.
- EXPOSE does not make the ports of the container accessible to the host. To do that, you must use either the -p flag to publish a range of ports or the -P flag to publish all of the exposed ports.

ENV

- **ENV** <key> <value>
- **ENV** <key>=<value> ...
- The ENV instruction sets the environment variable <key> to the value<value>. This value will be in the environment of all “descendent” Dockerfile commands and can be replaced inline in many as well.
- The ENV instruction has two forms. The first form, ENV <key> <value>, will set a single variable to a value. The entire string after the first space will be treated as the <value> - including characters such as spaces and quotes.

ENV

ENV myName="John Doe“ myDog=Rex\ The\
Dog \   myCat=fluffy

- And
  ENV myName John Doe
  ENV myDog Rex The Dog
  ENV myCat fluffy

will yield the same net results in the final container, but the first form is preferred because it produces a single cache layer.

ADD

- ADD <src>... <dest>
- ADD ["<src>",... "<dest>"] (required for paths containing whitespace)
- The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the container at the path <dest>.
- Multiple <src> resource may be specified but if they are files or directories then they must be relative to the source directory that is being built (the context of the build).

ADD

- The <dest> is an absolute path, or a path relative to WORKDIR, into which the source will be copied inside the destination container.
- **ADD** test relativeDir/
  - *# adds "test" to `WORKDIR`/relativeDir/*
- **ADD** test /absoluteDir
  - *# adds "test" to /absoluteDir*

www.fandsindia.com

COPY

- COPY <src>... <dest>
- COPY ["<src>",... "<dest>"] (required for paths containing whitespace)
- The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
- Multiple <src> resource may be specified but they must be relative to the source directory that is being built (the context of the build).

www.fandsindia.com

ENTRYPOINT

- ENTRYPOINT ["executable", "param1", "param2"] (exec form, preferred)
- ENTRYPOINT command param1 param2 (shell form)
- An ENTRYPOINT allows you to configure a container that will run as an executable.
- For example, the following will start nginx with its default content, listening on port 80:
  - docker run -i -t --rm -p 80:80 nginx
- Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.

VOLUME

- VOLUME ["/data"]
- The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers. The value can be a JSON array, VOLUME ["/var/log/"], or a plain string with multiple arguments, such as VOLUME /var/log or VOLUME /var/log /var/db

VOLUME

- The docker run command initializes the newly created volume with any data that exists at the specified location within the base image.
- **Note**: If any build steps change the data within the volume after it has been declared, those changes will be discarded.
- **Note**: The list is parsed as a JSON array, which means that you must use double-quotes (“) around words not single-quotes (‘).

USER

- USER daemon
- The USER instruction sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.

WORKDIR

- WORKDIR /path/to/workdir
- The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.
- It can be used multiple times in the one Dockerfile. If a relative path is provided, it will be relative to the path of the previous WORKDIR instruction

WORKDIR

WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
The output of the final pwd comman d would be /a/b/c.

- use environment variables explicitly set in theDockerfile.
- **ENV** DIRPATH /path **WORKDIR** $DIRPATH/$DIRNAME **RUN** pwd
- The output of the final pwd command in this Dockerfile would be/path/$DIRNAME

ARG

- ARG <name>[=<default value>]
- The ARG instruction defines a variable that users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag.
- If a user specifies a build argument that was not defined in the Dockerfile, the build outputs an error.
- One or more build-args were not consumed, failing build.

ARG

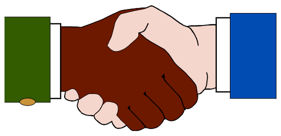
- Can define a single variable by specifying ARG once or many variables by specifying ARG more than once
- May optionally specify a default value for an ARG instruction
- An ARG variable definition comes into effect from the line on which it is defined in the Dockerfile not from the argument’s use on the command-line or elsewhere.

ARG vs ENV

- You can use an ARG or an ENV instruction to specify variables that are available to the RUN instruction.
- Environment variables defined using the ENV instruction always override an ARG instruction of the same name.
- ARG variables are not persisted into the built image as ENV variables are. However, ARG variables do impact the build cache in similar ways.

QUESTION / ANSWERS

THANKING YOU !



www.fandsindia.com