



**CLOUDBEES**  
**UNIVERSITY**

# CLOUDBEES JENKINS PLATFORM

Pipeline with Docker

# TOC

- Prerequisites
- Jenkins Overview
- CloudBees Introduction
- Pipeline Introduction
- Docker Introduction
- The Project (Part 1/2)
- The Project (Part 2/2)



# PREREQUISITES

## PREREQUISITES

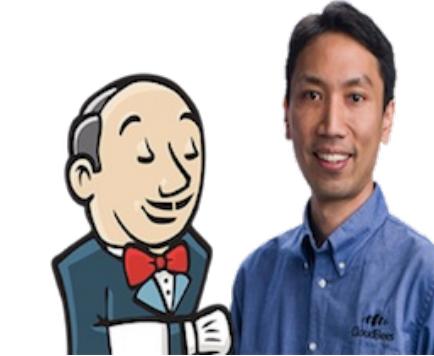
- If you do not already have an SSH client, please install [Secure Shell Chrome Extension](#)
- Confirm that you can login to the assigned server (IP has been sent by email). Use **cb** as both the user and the password.



# JENKINS OVERVIEW

# MEET JENKINS...

- #1 Continuous Integration and Delivery server
- Created by Kohsuke Kawaguchi
- An independent and active community ([jenkins-ci.org](http://jenkins-ci.org))
- 10 years old
- 500+ releases to date
- 100,000 active installations
- 300,000 Jenkins servers
- 1,200+ plugins



Source: 2014 Java Tools and Technologies Landscape – Rebel Labs



## THE 2014 LEADERBOARD OF JAVA TOOLS & TECHNOLOGIES

REBELLABS

82.5%  
JUnit\*

TOP TESTING  
FRAMEWORK  
USED BY  
DEVELOPERS

70%  
Jenkins®

MOST USED CI SERVER  
IN THE INDUSTRY

64%  
Maven

MOST USED  
BUILD TOOL  
IN JAVA

64%  
Nexus®

THE MAIN  
REPOSITORY  
USED BY  
DEVELOPERS

69% Git\*

#1 VERSION CONTROL  
TECHNOLOGY OUT THERE

67.5%  
Hibernate™

THE TOP  
ORM  
FRAMEWORK  
USED

65% Java 7

THE INDUSTRY LEADER FOR  
SE DEVELOPMENT

56%  
MongoDB®

THE NOSQL  
TECHNOLOGY OF CHOICE

55%  
FindBugs™

MOST-USED STATIC  
CODE ANALYSIS TOOL

50%  
Tomcat®

THE MOST POPULAR  
APPLICATION SERVER

49%  
Java EE 6

FOUND IN THE MOST  
ENTERPRISES

48%  
Eclipse

THE IDE USED MORE  
THAN ANY OTHER

40% Spring MVC™

MOST COMMONLY USED WEB FRAMEWORK

32% MySQL®

THE MOST POPULAR SQL TECHNOLOGY

\* Multiple selections possible

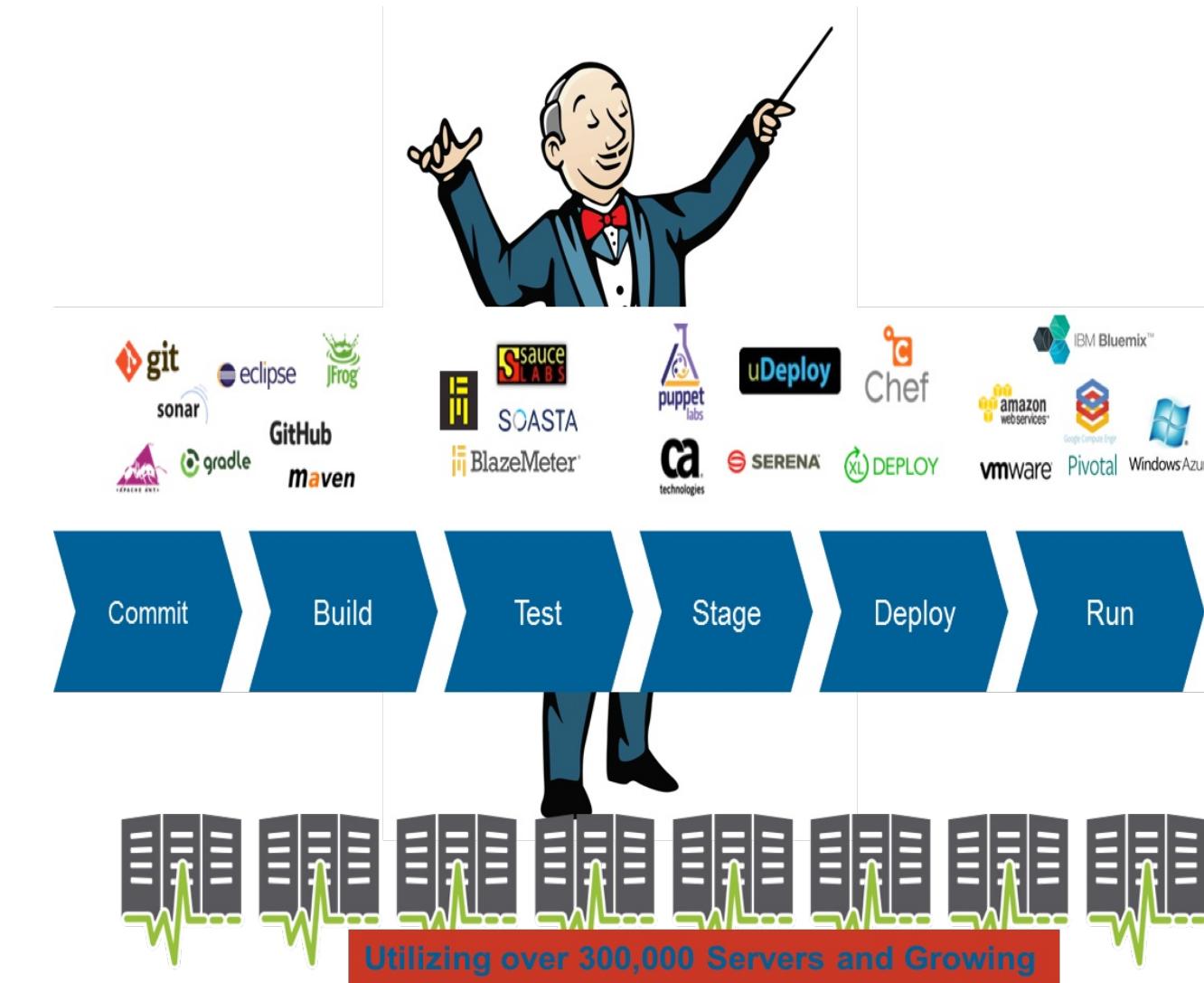
\* Normalized to exclude non-user base

Sample population of 2164 Java professionals, sample error 2.1%





# JENKINS IS THE CD ORCHESTRATOR





**CLOUDBEES  
UNIVERSITY**

---

# CLOUDBEES INTRODUCTION

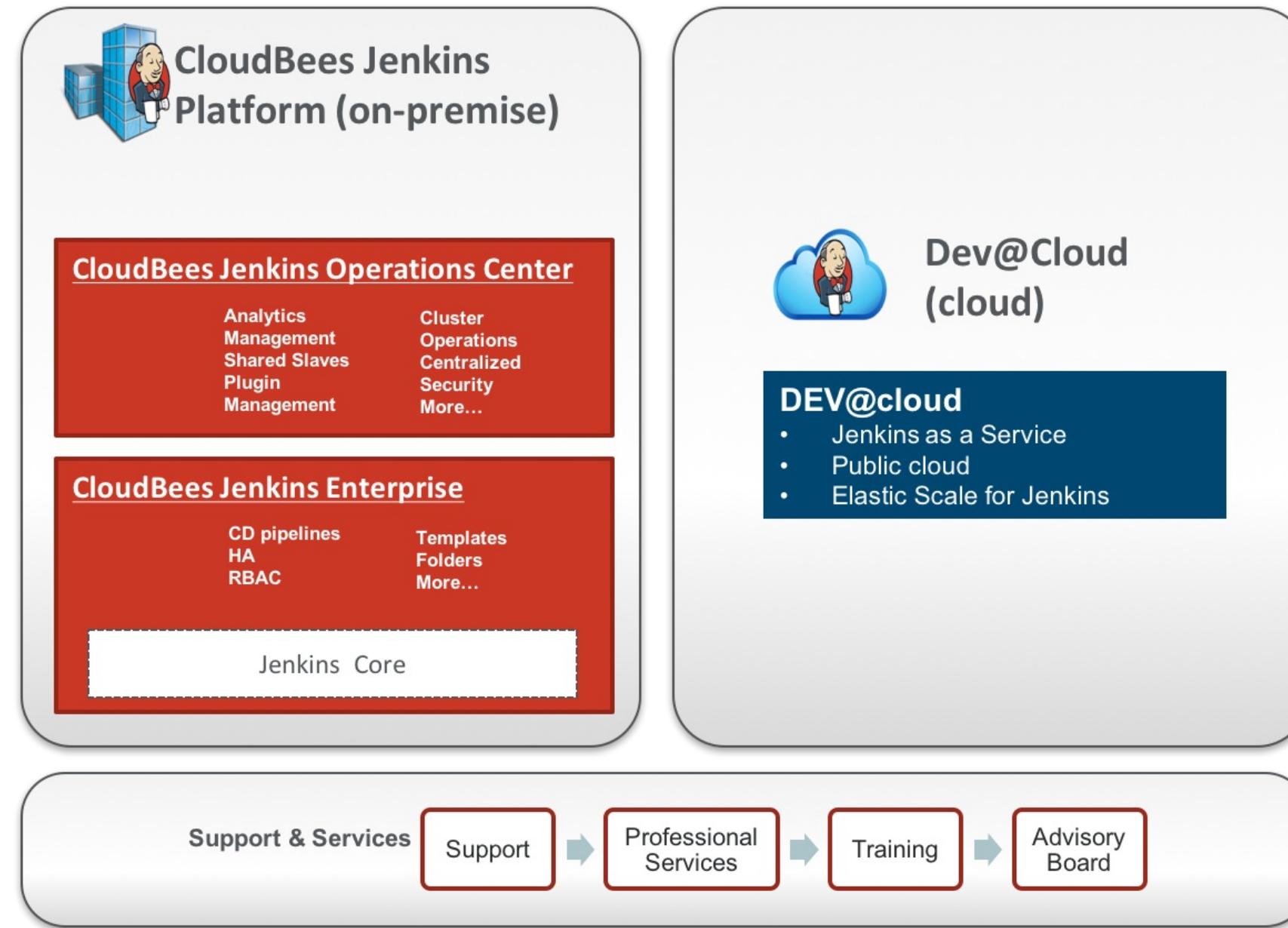
# CLOUDBEES AND THE JENKINS COMMUNITY

- Kohsuke Kawaguchi
  - Community leader and CloudBees' CTO
- Code and Releases
  - CloudBees partners with the community on development
  - CloudBees engineers contribute a majority of Jenkins OSS code
  - CloudBees partners with the community on releases
  - CloudBees contributes fixes back to the community
- Produce Jenkins Quarterly Newsletter
- Conduct Jenkins User Conferences





# CLOUDBEES PRODUCTS



# CLOUDBEES JENKINS PLATFORM: ENTERPRISE EDITION

For enterprise DevOps or infrastructure teams that want to:

- Have the best in class Jenkins technical support
- Operate Jenkins at scale to support a large number of developers, teams, or projects
- Monitor usage and share resources across the Jenkins infrastructure
- Optimize performance and efficiency of the CD platform

# CLOUDBEES SUBSCRIPTION MODEL

- CloudBees sells subscriptions that entitle you to receive support for:
  - CloudBees Jenkins Platform
  - OSS Jenkins including 1,200+ Open Source Plugins
- Customers are charged an annual subscription fee per installation
  - Multi-year subscriptions are available
- A typical service subscription includes:
  - Software updates, bug fixes, and upgrades
  - Technical support
  - stable versions, stable APIs, and more

# CLOUDBEES RESOURCES

- Customer Engagement
  - Support
  - Knowledge Base
  - Diagnostics
- Professional Services
  - Architecture Assessment & CD Guidance
  - Bootstrap Implementation Services
  - Migration Assistance
- Partners
  - Training
  - Integration
  - Custom Development





# PIPELINE INTRODUCTION (DAY 1)

## IN THIS UNIT: YOU WILL LEARN

- The need for pipeline
- Pipeline use cases
- Key pipeline DSL
- Pipeline structure and syntax
- Execution control

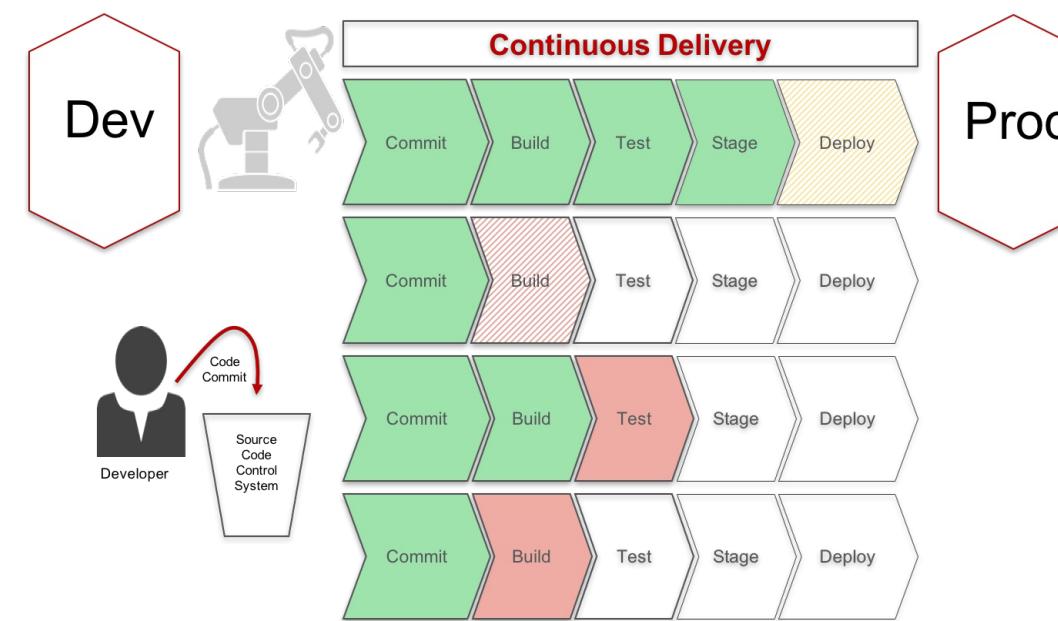
## IN THIS UNIT: YOU WILL BE ABLE TO

- Recognize pipeline purpose and use cases

# THE NEED FOR PIPELINE?

- Reduction of a number of jobs
- Easier maintenance
- Deployment decentralization
- Easier specification through code

# WHAT IS CD FLOW?



CD is about setting a "flow", from SCM commit to deployed application  
Jenkins can chain jobs this way, but ...

# JENKINS CD FLOW

A Real-world CD Flow Is Way More Complex !

- Requires (Complex) Conditional Logic
- Requires Resources Allocation And Cleanup
- Involves Human Interaction For Manual Approval
- Should Be Resumable At Some Point On Failure

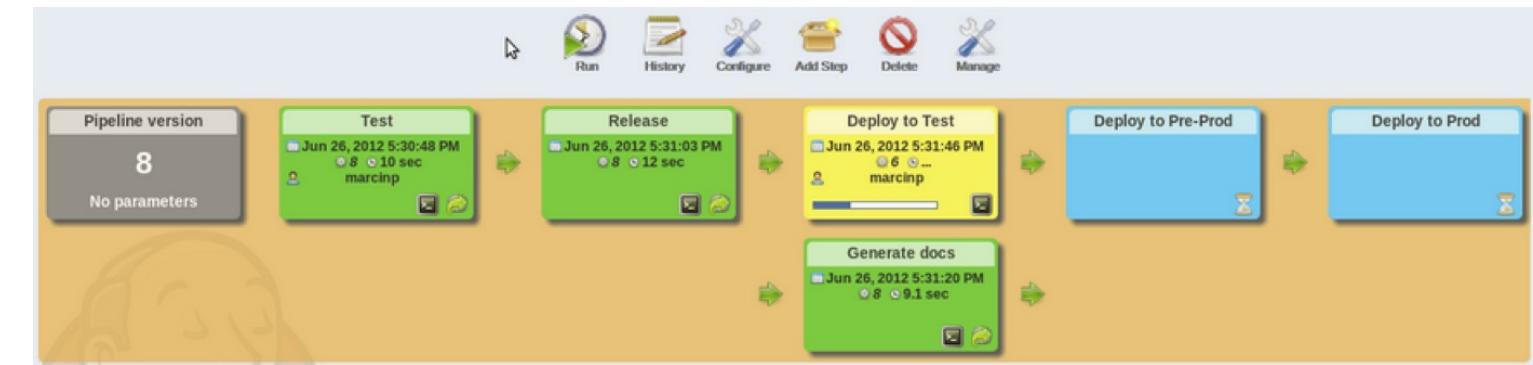
# JENKINS CD FLOW

- Many Related Plugins
  - Conditional build steps
  - Parameterized trigger
  - Promotions
  - ...
- Major Issue: Flow Configuration Is Scattered In Various Jobs

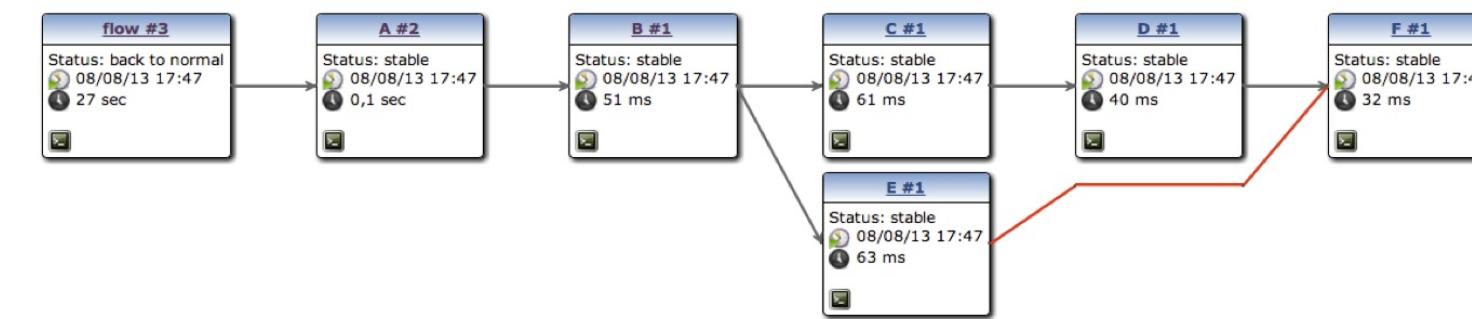


# CD FLOW VISUALIZATION

Build Pipeline Plugin To Render “Linear” Pipeline



Build-graph-view To Render Complex Ones

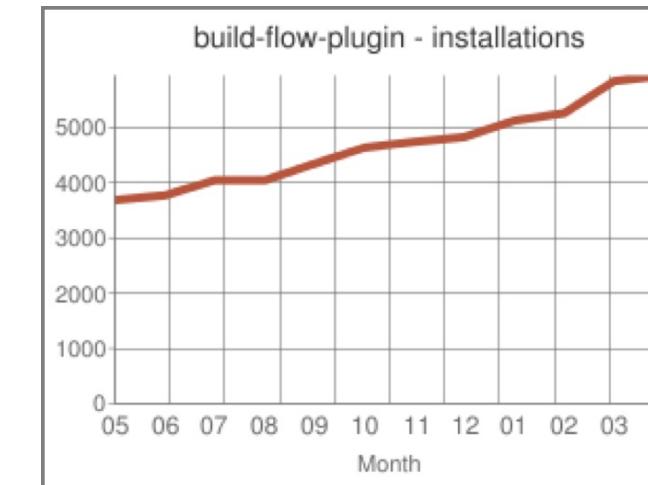


## REALITY CHECK

- How do you use Jenkins now?
- Given the functionality, where do you see your team using these tools and plugins in your jobs?
- Do you practice CI or CD?
- Describe your CI/CD flow needs and processes.

# BUILD FLOW PLUGIN

- OSS
- Proof of concept
- Define CD flow as code
- Largely adopted
- Technical limitations



# CLOUDBEES PIPELINE PLUGIN

- Inspired By Build-flow
- Groovy DSL To Orchestrate Build Steps
- Extensible DSL Syntax
- Supports Checkpoint
- Advanced Visualization
- SCM Friendly

## MID-BREAK

(10) minutes for learner re-integration.



# CREATING A SIMPLE PIPELINE

Item name

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

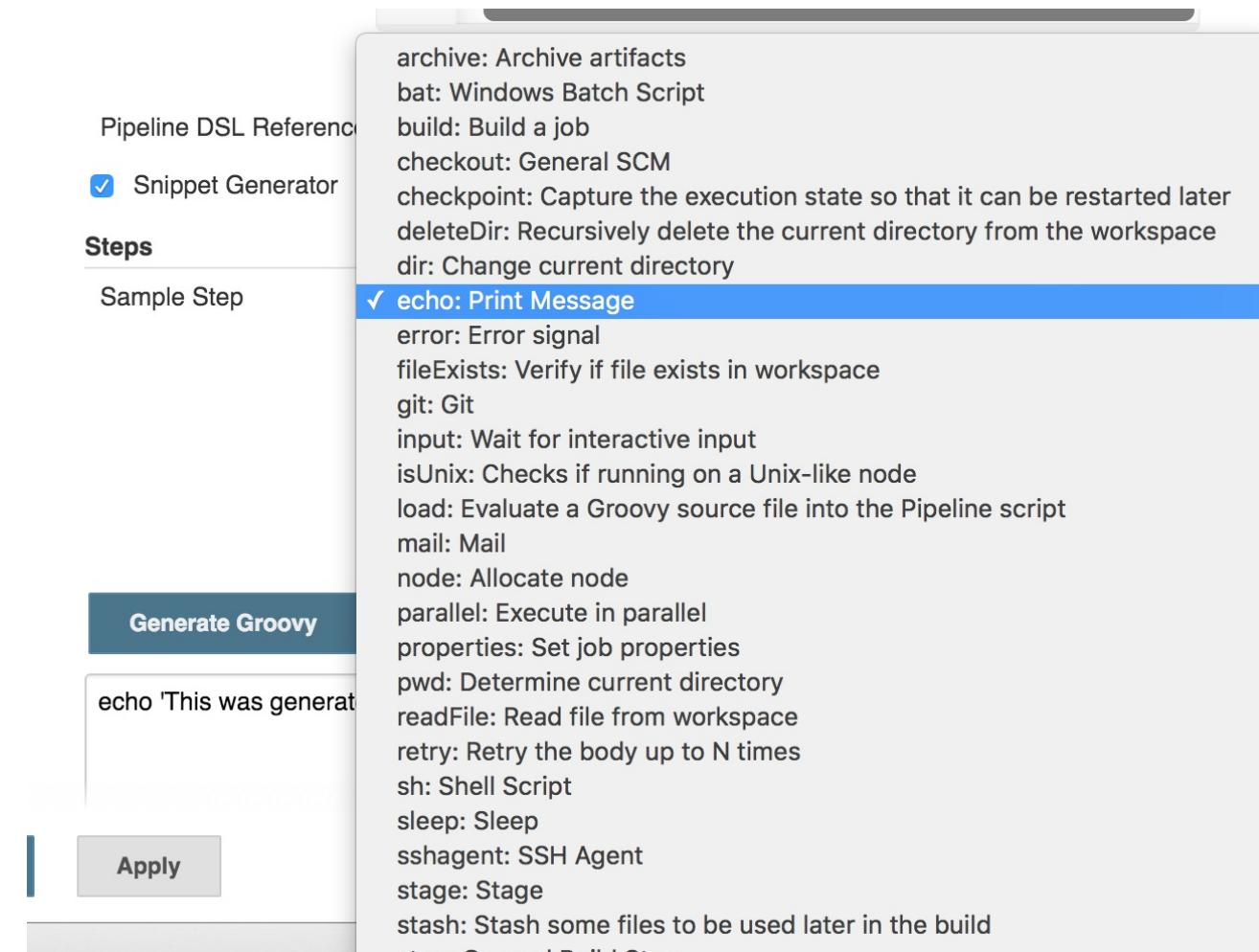
**Pipeline**  
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Auxiliary Template**  
Auxiliary templates are used to create nested structures embedded within other templates as attribute values. For example, if you are modeling a CD, you'd define an auxiliary template called "Track, and then your "CD" would have an attribute called "tracks" that contain a list of "Track"s.

**Backup**

More info: [Getting Started With Pipeline](#)

# SNIPPET GENERATOR



The screenshot shows a user interface for generating Jenkins Pipeline code. On the left, there's a sidebar with 'Pipeline DSL Reference' and a checked checkbox for 'Snippet Generator'. Below that is a 'Steps' section with a 'Sample Step' button. In the center, a modal window is open with a title bar 'Pipeline DSL Reference' and a 'Generate Groovy' button. The main area of the modal lists various Pipeline steps. One step, 'echo: Print Message', is highlighted with a blue background and a checkmark icon. Other listed steps include archive, bat, build, checkout, checkpoint, deleteDir, dir, error, fileExists, git, input, isUnix, load, mail, node, parallel, properties, pwd, readFile, retry, sh, sleep, sshagent, stage, stash, and stop.

- archive: Archive artifacts
- bat: Windows Batch Script
- build: Build a job
- checkout: General SCM
- checkpoint: Capture the execution state so that it can be restarted later
- deleteDir: Recursively delete the current directory from the workspace
- dir: Change current directory
- echo: Print Message**
- error: Error signal
- fileExists: Verify if file exists in workspace
- git: Git
- input: Wait for interactive input
- isUnix: Checks if running on a Unix-like node
- load: Evaluate a Groovy source file into the Pipeline script
- mail: Mail
- node: Allocate node
- parallel: Execute in parallel
- properties: Set job properties
- pwd: Determine current directory
- readFile: Read file from workspace
- retry: Retry the body up to N times
- sh: Shell Script
- sleep: Sleep
- sshagent: SSH Agent
- stage: Stage
- stash: Stash some files to be used later in the build
- stop: General Build Stop



# PIPELINE INTRODUCTION: REVIEW

# PIPELINE INTRODUCTION: REVIEW

- CD Flow
- Older Solutions
- Build Flow Plugin
- Cloudbees Pipeline Plugin
- Create Pipeline Jobs
- Use The Snippet Generator

# PIPELINE INTRODUCTION: EXERCISE

Pipeline Introduction: Exercise

# DOCKER INTRODUCTION (DAY 2)

# REVIEW OF DAY 1 CONCEPTS AND EXERCISE

- CD flow
- Older solutions
- Build Flow plugin
- CloudBees Pipeline Plugin
- Create Pipeline jobs & use the Snippet Generator
- Exercise

# IN THIS UNIT: YOU WILL LEARN

- How to prepare the environments
- Docker containers
- Docker use cases
- Docker tools

## IN THIS UNIT: YOU WILL BE ABLE TO

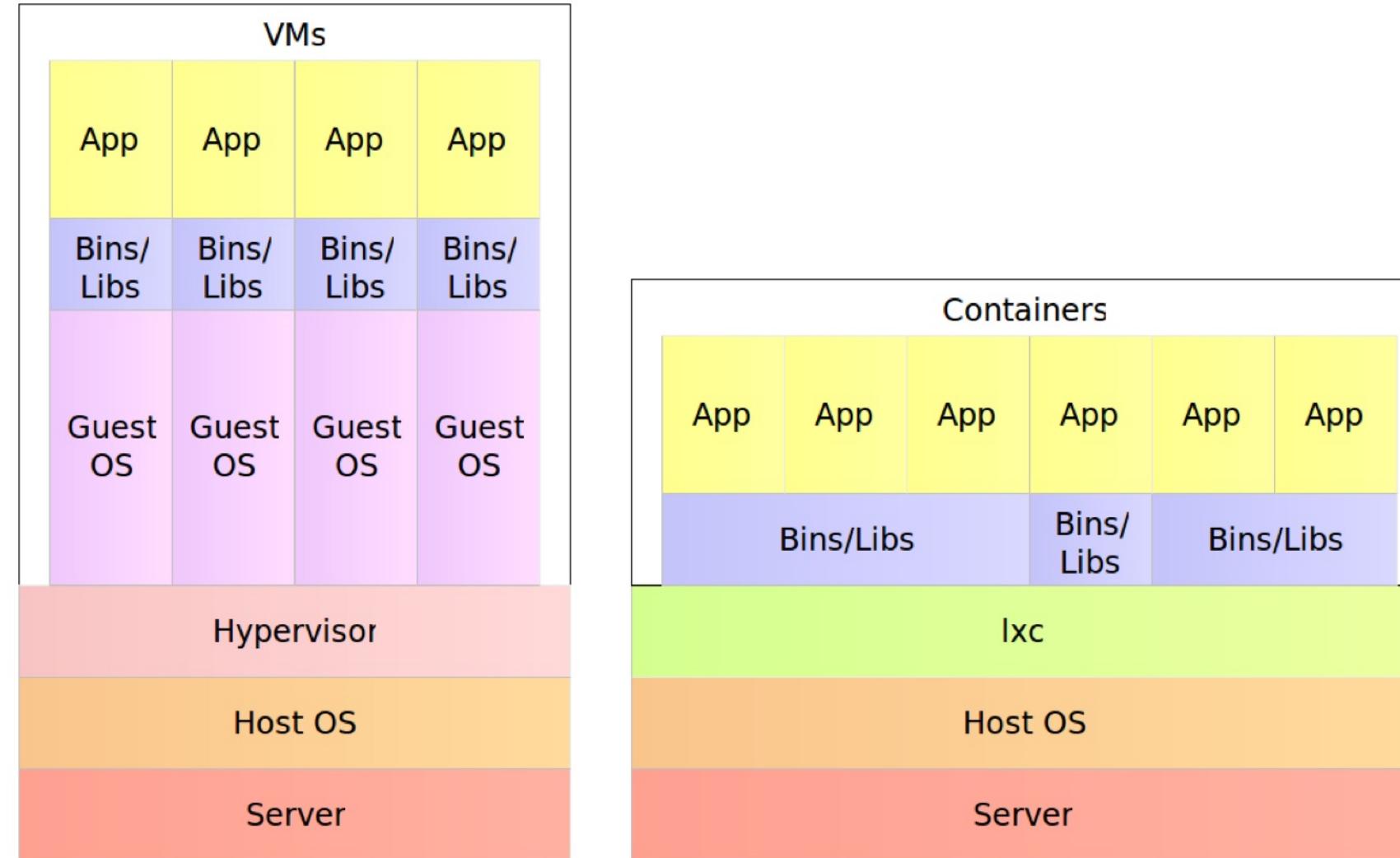
- Prepare the CD flow environments
- Understand Docker containers and use cases
- Have A Working Knowledge Of Docker Tools

# DOCKER CONTAINERS

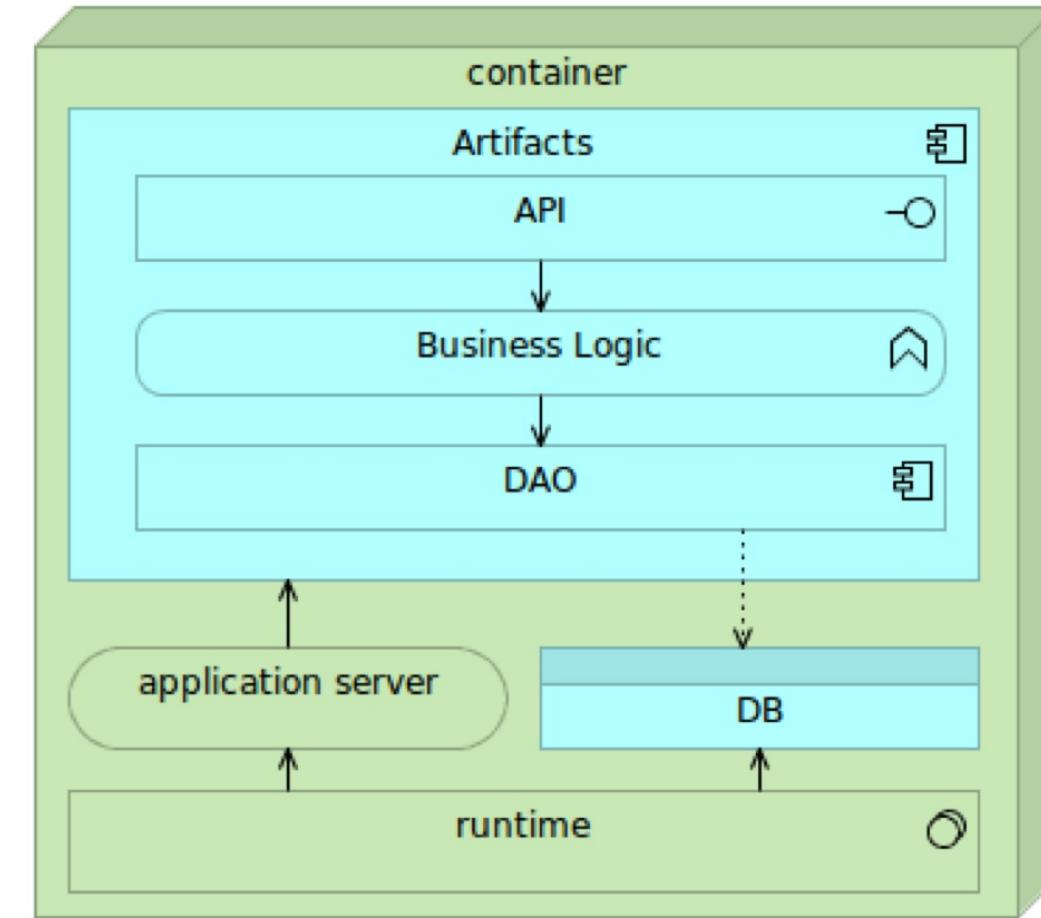
- Self-sufficient
- Isolated
- Immutable
- Reliable
- Scalable



# VMS VS DOCKER CONTAINERS

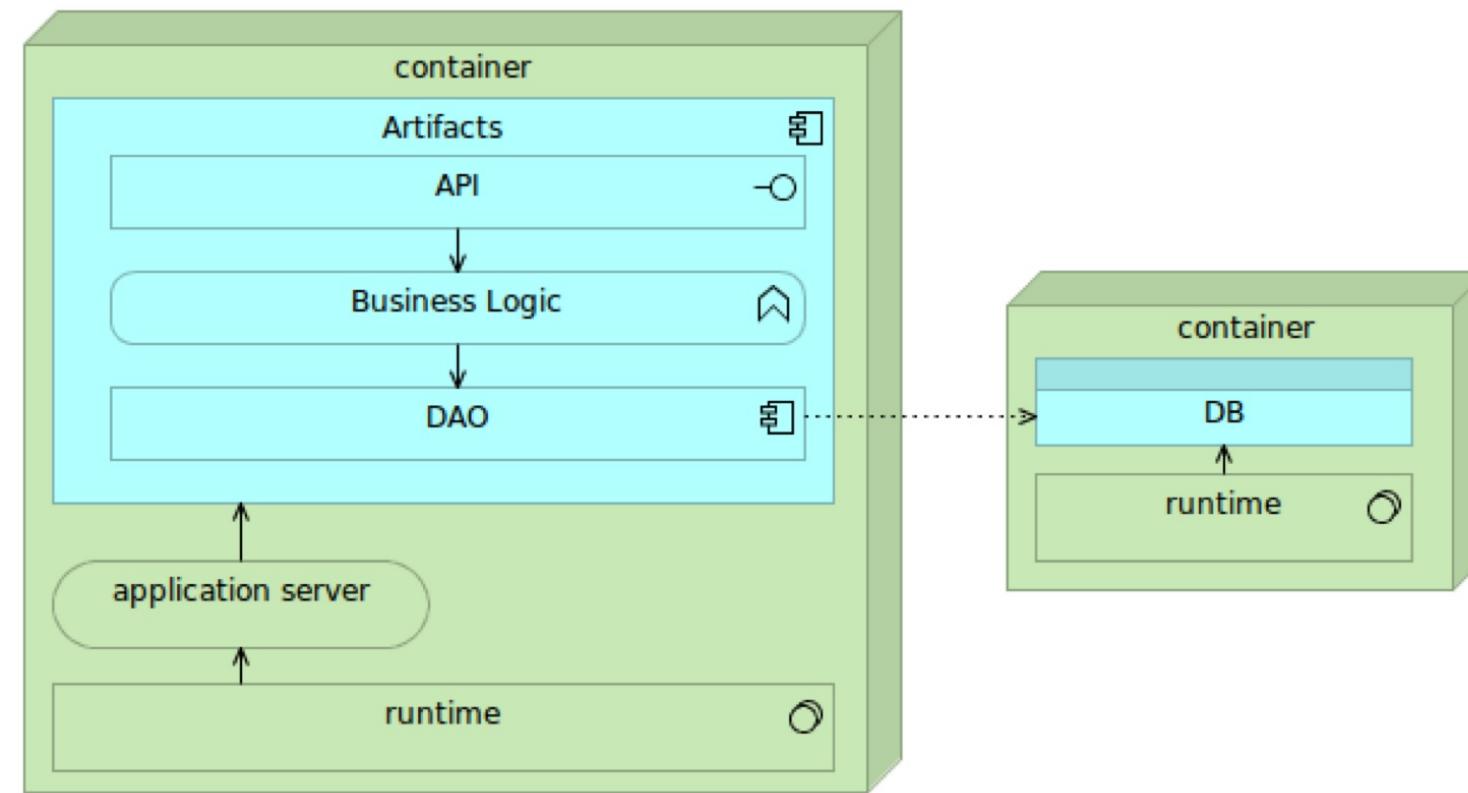


# SINGLE-CONTAINER SERVICE





# LINKED OR NETWORKED CONTAINERS



# DOCKER USE CASES

- Local Development And Testing
- Continuous Integration, Delivery, Or Deployment
- Reliable And Repeatable Deployments

## REALITY CHECK

- Questions About Containers?
- Examples Of Scalability In Your Current Use Of Containers?
- How Is CI Valuable In Your Environments?
- Are You Using Containers In Deployment Now?

# DOCKER TOOLS

- Docker Hub/Registry
- Docker Engine
- Docker Compose
- Docker Swarm
- Docker Machine
- Kitematic

# DOCKER HUB AND REGISTRY



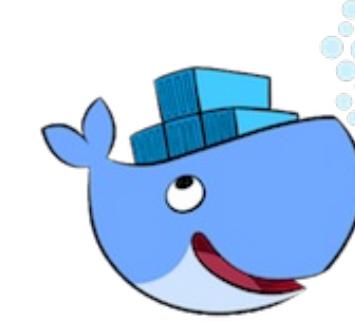
- **Docker Hub:** Cloud hosted service that provides registry capabilities for public and private content.
- **Docker Registry:** An open source application dedicated to the storage and distribution of your Docker images.
- **Docker Trusted Registry:** Allows storing and managing Docker images on-premise. It supports security or regulatory compliance requirements.

# DOCKER HUB AND REGISTRY: PULLING AND PUSHING CONTAINERS

Please use the SSH client to connect to the server you were assigned.

```
docker pull cloudbees/training-books-ms-tests  
docker tag cloudbees/training-books-ms-tests \  
      localhost:5000/training-books-ms-tests  
docker push localhost:5000/training-books-ms-tests
```

# DOCKER ENGINE



- **Docker Engine** is a lightweight runtime and robust tooling that builds and runs Docker containers.

# DOCKER ENGINE: RUNNING TESTS INSIDE CONTAINERS

```
cd /mnt/training-books-ms

docker run -it --rm \
-v $PWD/client/components:/source/client/components \
-v $PWD/client/test:/source/client/test \
-v $PWD/src:/source/src \
-v $PWD/target/scala-2.10:/source/target/scala-2.10 \
--env TEST_TYPE=all \
localhost:5000/training-books-ms-tests

ll target/scala-2.10/
docker ps -a | grep books
```

# DOCKER ENGINE: BUILDING CONTAINERS AND PUSHING THEM TO THE PRIVATE REGISTRY

```
cat Dockerfile  
docker build -t localhost:5000/training-books-ms .  
docker push localhost:5000/training-books-ms
```

## MID-BREAK

(10) minutes for learner re-integration.



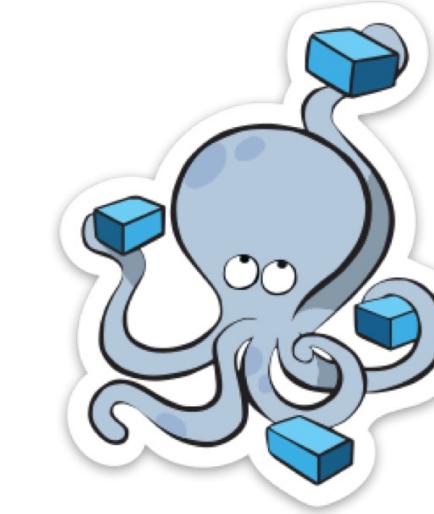
# DOCKER ENGINE: RUNNING CONTAINERS

```
docker run -d --name books-ms-db mongo  
  
docker run -d --name books-ms \  
  -p 1234:8080 \  
  --link books-ms-db:db \  
  localhost:5000/training-books-ms  
  
docker exec -it books-ms env | grep DB  
  
docker ps  
  
curl -I localhost:1234/api/v1/books
```

# DOCKER ENGINE: LOGGING, STOPPING AND REMOVING CONTAINERS

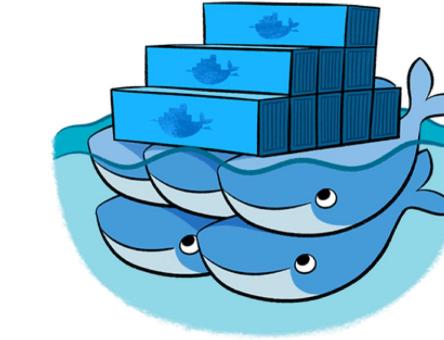
```
docker logs books-ms
docker logs books-ms-db
docker stop books-ms-db books-ms
docker ps -a
docker rm books-ms-db books-ms
docker ps -a
```

# DOCKER COMPOSE



- **Docker Compose** allows defining multi-container application with all of its dependencies in a single file, then spin your application up in a single command.

# DOCKER SWARM



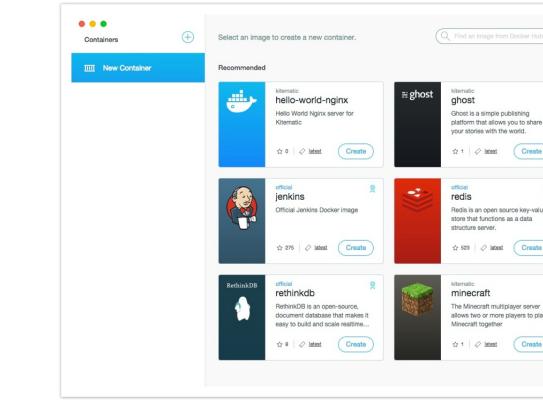
- **Docker Swarm** provides native clustering capabilities to turn a group of Docker engines into a single, virtual Docker Engine. With these pooled resources, you can scale out your application as if it were running on a single, huge computer.

# DOCKER MACHINE



- **Docker Machine** automatically sets up Docker on your computer, on cloud providers, and inside your data center. Docker Machine provisions the hosts, installs Docker Engine on them, and then configures the Docker client to talk to the Docker Engines.

# KITEMATIC



- **Kitematic** is a completely automated process that installs and configures the Docker environment on your machine. Build and run containers through a simple, yet powerful graphical user interface (GUI).

# DOCKER INTRODUCTION: REVIEW

# DOCKER INTRODUCTION: REVIEW

- Docker benefits & advantages
- Docker use cases
- Docker Hub and Registry, Engine, Compose, Swarm, Machine, Kitematic

# DOCKER INTRODUCTION: EXERCISE

Docker Introduction: Exercise



# THE PROJECT - PART 1 (DAY 3)

# REVIEW OF DAY 2 CONCEPTS AND EXERCISE

- Docker benefits & advantages
- Docker use cases
- Docker Hub and Registry, Engine, Compose, Swarm, Machine, Kitematic

## IN THIS UNIT: YOU WILL LEARN

- How to combine CloudBees Pipeline plugin with Docker
- How to implement most commonly used steps required for CI/CD flow

## IN THIS UNIT: YOU WILL BE ABLE TO

- Create deployment lifecycle with Jenkins Pipeline and Docker

# THE PROJECT

- Run pre-deployment tests inside a Docker container
- Build artefacts
- Build and push the service container
- Request manual permission to deploy the service container to production

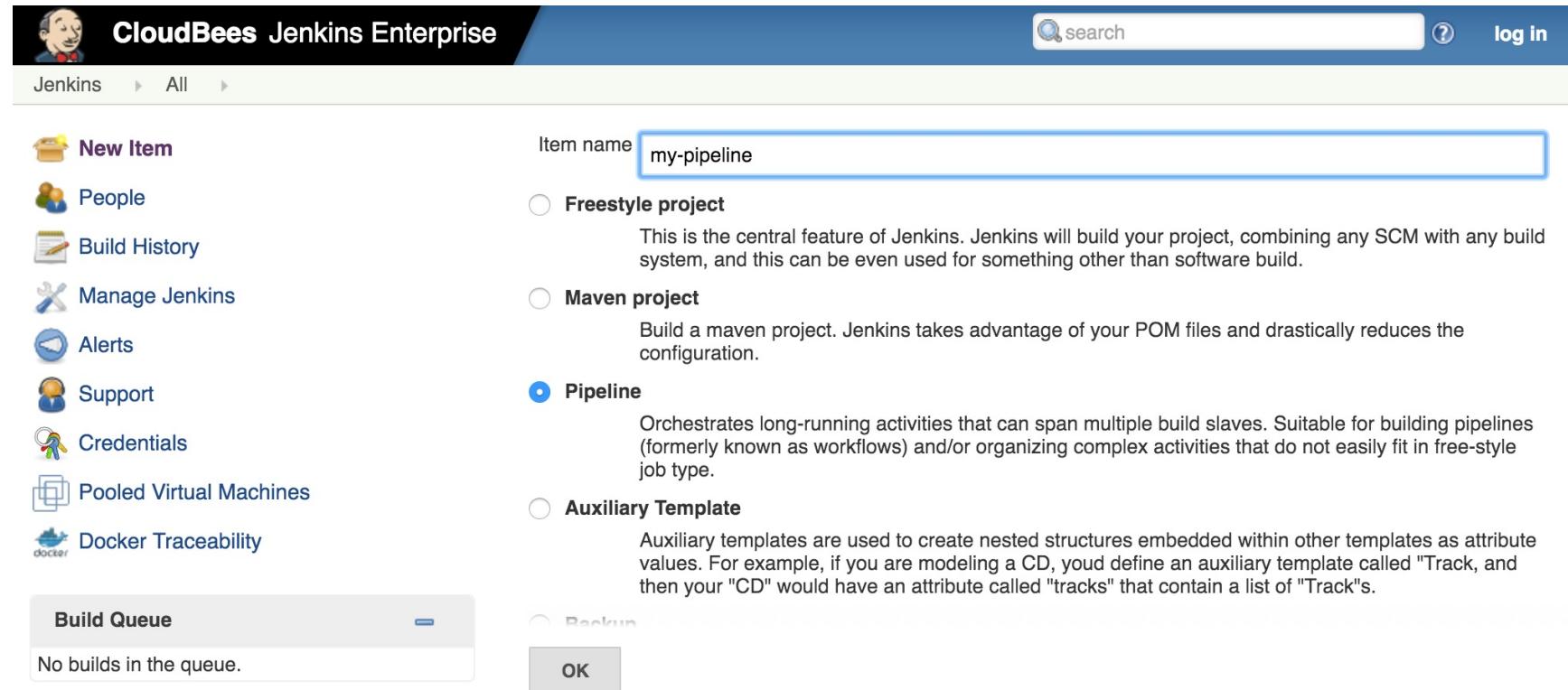
## THE PROJECT (CONT.)

- Pull the service container and all dependent containers to production
- Run the service container and all dependent containers in production
- Run post-deployment tests (integration tests) inside a Docker container

## REALITY CHECK

- Questions on the preparation and the project?
- Does this workflow differ from your practices?
- Using containers now?

# CREATE A PIPELINE JOB CALLED MY-PIPELINE



The screenshot shows the Jenkins dashboard with the title "CloudBees Jenkins Enterprise". On the left, there is a sidebar with various links: "New Item", "People", "Build History", "Manage Jenkins", "Alerts", "Support", "Credentials", "Pooled Virtual Machines", and "Docker Traceability". Below this is a "Build Queue" section stating "No builds in the queue.". The main content area has a search bar and a "log in" button. A form is open for creating a new item, with the "Item name" field containing "my-pipeline". There are five project type options: "Freestyle project", "Maven project", "Pipeline" (which is selected), "Auxiliary Template", and "Backup". Each option has a detailed description below it. At the bottom right of the form is an "OK" button.

## KEY PIPELINE DSL – NODE: TASK

Specify the node `cd`, run the pipeline, and confirm that it is running inside the `cd` node by looking at logs.

# KEY PIPELINE DSL – NODE: SOLUTION

```
node("cd") {  
}
```

## RUN THE JOB

Open <http://<IP>:8080/job/my-pipeline/build?delay=0sec>

Open <http://<IP>:8080/job/my-pipeline/lastBuild/console>



### Console Output

```
Started by user anonymous
[Workflow] Allocate node : Start
Running on node-cd in /data/jenkins_slave/workspace/my-workflow
[Workflow] node {
[Workflow] } //node
[Workflow] Allocate node : End
[Workflow] End of Workflow
Finished: SUCCESS
```

## KEY PIPELINE DSL – GIT: TASK

Clone the code from the repository <https://github.com/cloudbees/training-books-ms.git>

# KEY PIPELINE DSL – GIT: SOLUTION

```
git "https://github.com/cloudbees/training-books-ms.git"
```

# KEY PIPELINE DSL – VARIABLES, PWD AND SH: TASK

Assign the current job workspace directory to the **dir** variable, create directory **db** inside the workspace, and assign full permissions to all users.

# KEY PIPELINE DSL – VARIABLES, PWD AND SH: SOLUTION

```
def dir = pwd()  
sh "mkdir -p ${dir}/db"  
sh "chmod 0777 ${dir}/db"
```

# KEY PIPELINE DSL – STAGE: TASK

Create the **pre-deployment tests** stage.

# KEY PIPELINE DSL – STAGE: SOLUTION

```
stage "pre-deployment tests"
```

## MID-BREAK

(10) minutes for learner re-integration.



## KEY PIPELINE DSL – DOCKER: TASK

Pull the Docker image `localhost:5000/books-ms-tests` and run the `run_tests.sh` script inside the container. Host volume `db` should be mounted as `/data/db` inside the container.

# KEY PIPELINE DSL – DOCKER: SOLUTION

```
def tests = docker.image("localhost:5000/training-books-ms-tests")
tests.pull()
tests.inside("-v ${dir}/db:/data/db") {
    sh "./run_tests.sh"
}
```

## KEY PIPELINE DSL – DOCKER: TASK

Build the Docker image **localhost:5000/books-ms** and push the container to the private registry.  
Use the stage **build** for these steps.

# KEY PIPELINE DSL – DOCKER: SOLUTION

```
stage "build"
def service = docker.build "localhost:5000/training-books-ms"
service.push()
```

## KEY PIPELINE DSL – STASH AND UNSTASH: TASK

Pull containers and run (through Docker Compose target `app`) the `localhost:5000/books-ms` container in the `production` node. Use `stash` to archive `docker-compose-dev.yml` file while in the `cd` node and `unstash` to retrieve it when inside the `production` node. Before running the service, make sure that both the service and mongo containers are pulled.

# KEY PIPELINE DSL – STASH AND UNSTASH: SOLUTION

```
node("cd") {  
    ...  
    stash includes: "docker-compose*.yml", name: "docker-compose"  
}  
node("production") {  
    stage "deploy"  
    unstash "docker-compose"  
    docker.image("localhost:5000/training-books-ms").pull()  
    docker.image("mongo").pull()  
    sh "docker-compose -p books-ms up -d app"  
}
```

## KEY PIPELINE DSL – ENV AND WITHENV: TASK

Run post-deployment tests in the node `cd`. The `run_tests.sh` script expects two environment variables: `TEST_TYPE=integ` and `DOMAIN=<IP>:8081`. Tests should be run inside the `training-books-ms` container.

# KEY PIPELINE DSL – ENV AND WITHENV: SOLUTION

```
node("cd") {  
    stage "post-deployment tests"  
    def tests = docker.image("localhost:5000/training-books-ms-tests")  
    tests.inside() {  
        withEnv(["TEST_TYPE=integ", "DOMAIN=http://[IP]:8081"]) {  
            sh "./run_tests.sh"  
        }  
    }  
}
```

# THE PROJECT - PART 1: REVIEW

# THE PROJECT - PART 1: REVIEW

- Defined the project steps
- Created a new Pipeline job
- Defined steps that clone the code
- Defined steps that run pre-deployment tests
- Defined steps that build and push the container
- Defined steps that deploy the container
- Defined steps that run post-deployment tests

# THE PROJECT - PART 1: EXERCISE

The Project - Part 1: Exercise



# THE PROJECT - PART 2 (DAY 4)

## REVIEW OF DAY 3

- Defined the project steps
- Created a new Pipeline job
- Defined steps that clone the code
- Defined steps that run pre-deployment tests
- Defined steps that build and push the container
- Defined steps that deploy the container
- Defined steps that run post-deployment tests

## IN THIS UNIT: YOU WILL LEARN

- How to use more advanced features of the CloudBees Pipeline plugin and Docker

## IN THIS UNIT: YOU WILL BE ABLE TO

- Create advanced deployment lifecycle with Jenkins Pipeline and Docker

## THE PROJECT (CONT.)

- Request manual confirmation before deployment
- Run steps in parallel
- Control the steps execution
- Add checkpoints on critical or after long running processes
- Visualize the deployment pipeline
- Convert the job into a template
- Leverage multi branch pipeline and Jenkinsfile features

## KEY PIPELINE DSL – INPUT: TASK

- Add an input with the message **Please confirm deployment to production** and the button with the text **Submit**.
- The input should be requested prior to deployment to production.
- Add the parameter for additional notes and echo the value.
- Make sure that only the user **manager** is allowed to use this input.
- Use user/pass **manager/manager** to login before approving the deployment.

# KEY PIPELINE DSL – INPUT: SOLUTION

```
def response = input message: 'Please confirm deployment to production',
    ok: 'Submit',
    parameters: [
        $class: 'StringParameterDefinition',
        defaultValue: '',
        description: 'Additional comments',
        name: ''
    ],
    submitter: 'manager'
echo response
```

## KEY PIPELINE DSL – PARALLEL: TASK

Modify the script so that **service** and **db** containers are pulled in parallel.

# KEY PIPELINE DSL – PARALLEL: SOLUTION

```
//    docker.image("localhost:5000/training-books-ms").pull()
//    docker.image("mongo").pull()
def pull = [:]
pull["service"] = {
    docker.image("localhost:5000/training-books-ms").pull()
}
pull["db"] = {
    docker.image("mongo").pull()
}
parallel pull
```

## REALITY CHECK

- Questions on the prep and the Project?
- Clear on the input and parallel DSL?
- Ready for the execution control?

# KEY PIPELINE DSL – EXECUTION CONTROL

```
retry(10) {  
    // some block  
}  
  
sleep time: 10, unit: 'MINUTES'  
  
waitUntil {  
    // some block  
}  
  
timeout(time: 100, unit: 'SECONDS') {  
    // some block  
}  
  
while(something) {  
    // do something  
    if (something_else) {  
        // do something else  
    }  
}
```

# KEY PIPELINE DSL – EXECUTION CONTROL: TASK

Change the script so that the execution of **post-deployment tests** is retried in case of a failure.

# KEY PIPELINE DSL – EXECUTION CONTROL: SOLUTION

```
retry(2) {  
    sh "./run_tests.sh"  
}
```

# KEY PIPELINE DSL – EXECUTION CONTROL: TASK

Change the script so that there is a pause of **2 seconds** after the service is deployed.

# KEY PIPELINE DSL – EXECUTION CONTROL: SOLUTION

```
sleep 2
```

## MID-BREAK

(10) minutes for learner re-integration.



## KEY PIPELINE DSL – CHECKPOINT: TASK

Add checkpoint **deploy** between the nodes **cd** and **production**.

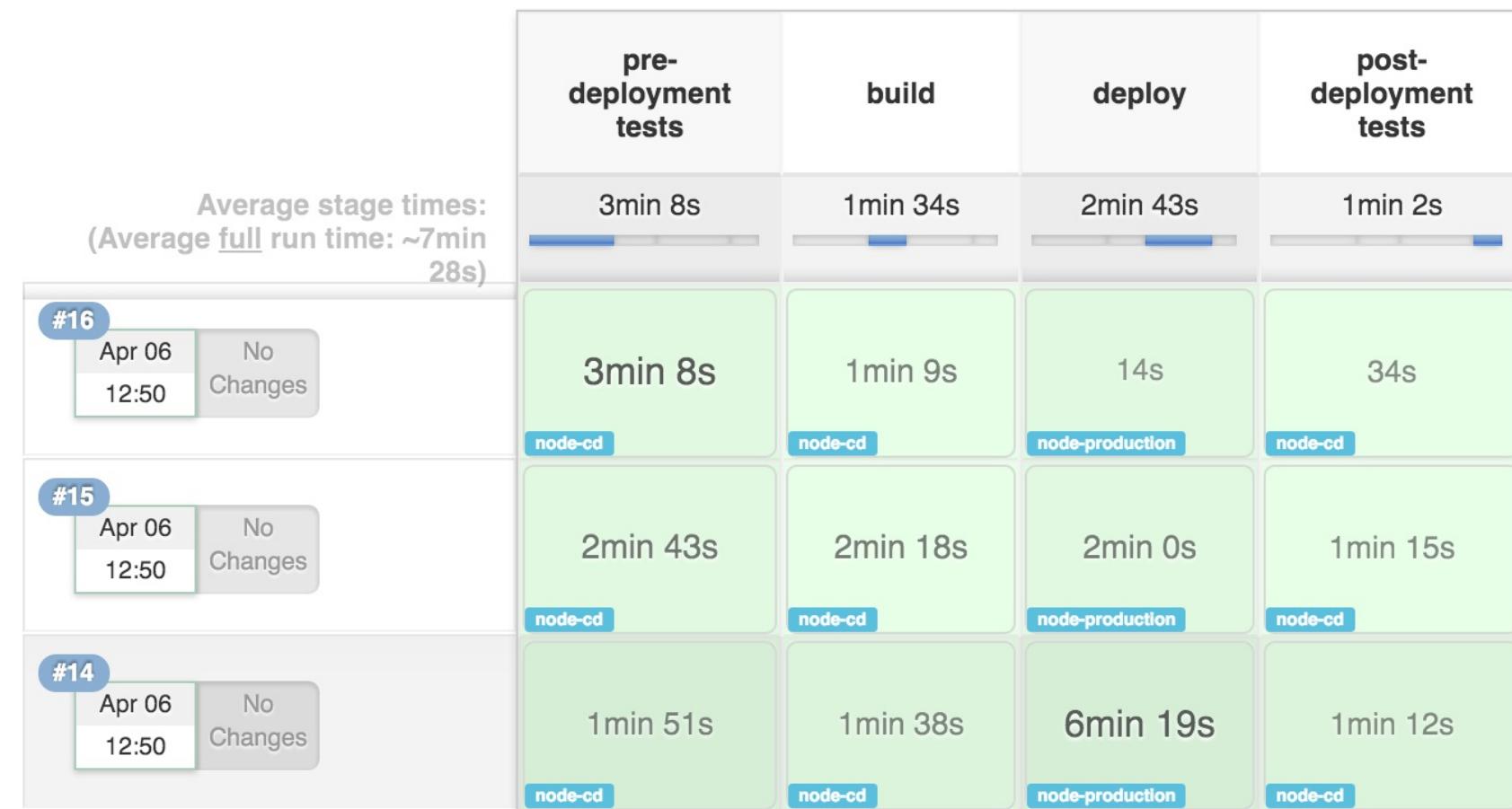
# KEY PIPELINE DSL – CHECKPOINT: SOLUTION

```
checkpoint "deploy"
```



# PIPELINE STAGE VIEW

## my-pipeline - Stage View



# PIPELINE AND JOB TEMPLATE: TASK

- Convert the deployment pipeline script into a template named **my-template**.
- Replace the service name (**training-books-ms**), the registry IP and port (**localhost:5000**) and domain (**http://<IP>:8081**) with variables.
- Create a new job called **my-pipeline-from-template**. The job type should be **my-template**.

# PIPELINE AND JOB TEMPLATE: SOLUTION

Name	<input type="text" value="my-job-from-template"/>
Service Name	<input type="text" value="training-books-ms"/>
Registry IP and Port	<input type="text" value="localhost:5000"/>
Domain	<input type="text" value="54.93.170.250"/>

**Save** **Apply**

# MULTI-BRANCH PIPELINE AND JENKINSFILE

- Multi-configuration project**

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

- Multibranch Pipeline**

Creates a set of Pipeline projects according to detected branches in one SCM repository.

- Publisher Template**

The publisher template lets you define a custom publisher by defining a number of attributes and describing how it translates to the configuration of another existing publisher. This allows you to create a locked down version of a publisher. It also lets you change the definition of the translation without redoing all the use of the template.

- My Job Template**

# MULTI-BRANCH PIPELINE AND JENKINSFILE

**Branch Sources**

 **Git**

Project Repository

Credentials   **Add**

Ignore on push notifications

**Advanced...**

Property strategy

**Add property** ▾

**Delete source**

# MULTI-BRANCH PIPELINE AND JENKINSFILE

```
def serviceName = "training-books-ms"
def registry = "localhost:5000"
def flow

node("cd") {
    git "https://github.com/cloudbees/${serviceName}.git"
    flow = load "/mnt/scripts/pipeline-common.groovy"
    flow.runPreDeploymentTests(serviceName, registry)
    flow.build(serviceName, registry)
}
checkpoint "deploy"
node("cd") {
    flow.deploy(serviceName, registry)
    flow.runPostDeploymentTests(serviceName, registry, "http://<IP>:8081")
}
```

# MULTI-BRANCH PIPELINE AND JENKINSFILE

```
[Workflow] Allocate node : End
[Workflow] } //node
[Workflow] Allocate node : End
[Workflow] End of Workflow
org.jenkinsci.plugins.scriptsecurity.sandbox.RejectedAccessException: Scripts not permitted to use
method groovy.lang.GroovyObject invokeMethod java.lang.String java.lang.Object
(org.jenkinsci.plugins.workflow.cps.CpsClosure2 sleep java.lang.Integer)
    at
org.jenkinsci.plugins.scriptsecurity.sandbox.whitelists.StaticWhitelist.rejectMethod(StaticWhitelist
.java:155)
    at
org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SandboxInterceptor.onMethodCall(SandboxIntercept
or.java:77)
    at
org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SandboxInterceptor.onMethodCall(SandboxIntercept
or.java:68)
    at org.kohsuke.groovy.sandbox.impl.Checker$1.call(Checker.java:149)
    at org.kohsuke.groovy.sandbox.impl.Checker.checkedCall(Checker.java:146)
    at com.cloudbees.groovy.cps.sandbox.SandboxInvoker.methodCall(SandboxInvoker.java:15)
    at Script1.deploy(Script1.groovy:36)
    at Unknown.Unknown(Unknown)
    at __cps.transform__(Native Method)
    at com.cloudbees.groovy.cps.impl.ContinuationGroup.methodCall(ContinuationGroup.java:69)
    at
```

# MULTI-BRANCH PIPELINE AND JENKINSFILE

No pending script approvals.

You can also remove all previous script approvals: [Clear Approvals](#)

---

Signatures already approved:

```
method groovy.lang.GroovyObject invokeMethod java.lang.String java.lang.Object
```

# PIPELINE GLOBAL LIBRARY

- Ability to share common parts of Pipeline scripts across multiple jobs
- Keep scripts DRY
- Shared library script Git repository
- Pipeline step **load** vs global library

Please visit Pipeline Global Library (<https://github.com/jenkinsci/workflow-cps-global-lib-plugin/blob/master/README.md>) for more info.

# THE PROJECT - PART 2: REVIEW

# THE PROJECT - PART 2: REVIEW

- Pipeline Job
- Job Template
- Multibranch Pipeline And Jenkinsfile

# COURSE REVIEW

# COURSE REVIEW

- The Need For The Pipeline
- What Is Cloudbees Pipeline?
- The Syntax And The Snippet Generator
- Docker Containers
- Docker Tools
- The Project

# THE PROJECT - PART 2: EXERCISE

The Project - Part 2: Exercise

# SYLLABUS AND REFERENCES

- CloudBees Pipeline Plugin Suite (<http://documentation.cloudbees.com/docs/cje-user-guide/workflow.html>)
- CloudBees Docker Pipeline Plugin (<http://documentation.cloudbees.com/docs/cje-user-guide/docker-workflow.html>)
- Docker Docs (<https://docs.docker.com/>)