CLOUDBEES UNIVERSITY

CLOUDBEES JENKINS PLATFORM

Pipeline with Docker (Day 3)

TOC

CLOUDBEES UNIVERSITY

THE PROJECT - PART 1 (DAY 3)

TOC

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

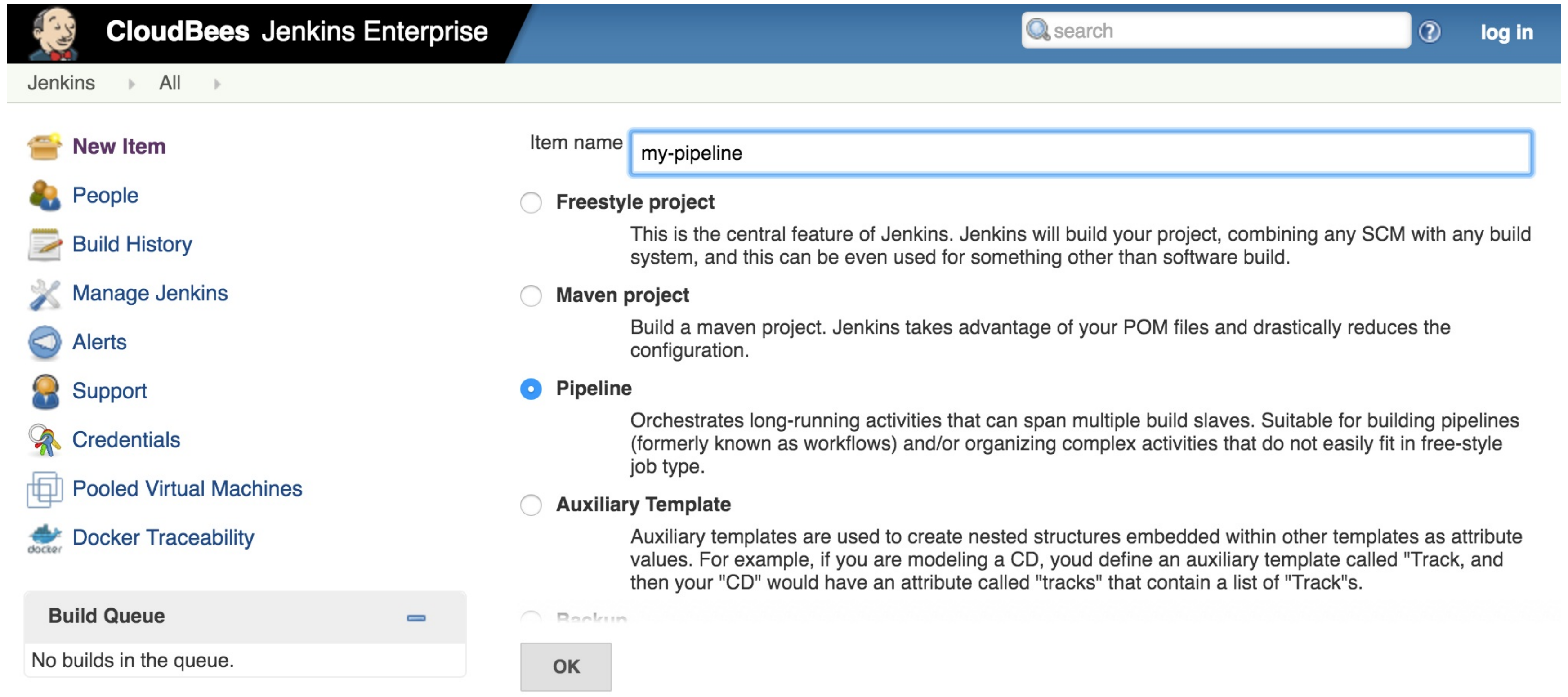
CLOUDBEES UNIVERSITY

REALITY CHECK

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

TOC

CREATE A PIPELINE JOB CALLED MY-PIPELINE



No builds in the queue.

Item name: my-pipeline

( ) **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, youd define an auxiliary template called "Track, and then your "CD" would have an attribute called "tracks" that contain a list of "Track"s.

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.

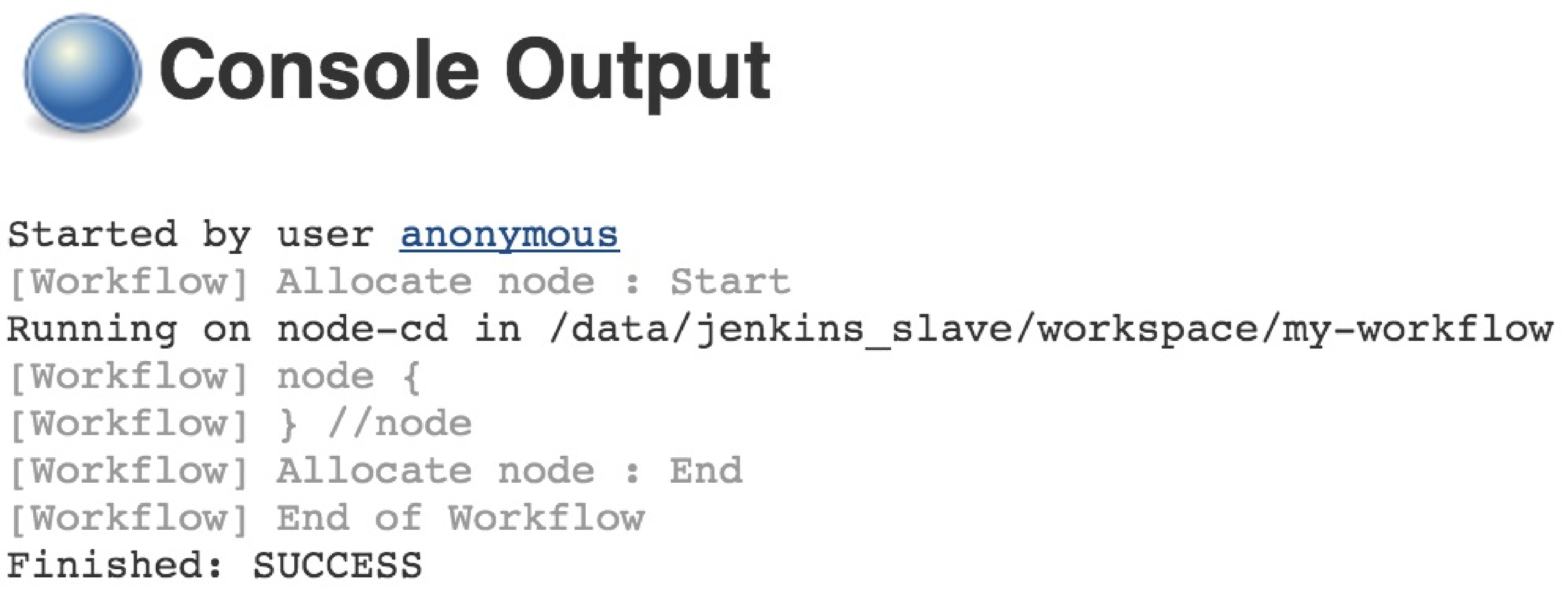
KEY PIPELINE DSL – NODE: SOLUTION

```
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