CLOUDBEES UNIVERSITY

CloudBees Jenkins Platform: Pipeline with Docker

Labs

Table of Contents

Introduction

This workbook is designed to supplement your CloudBees Jenkins training. It consists of a sequence of lab exercises that will introduce Jenkins Pipeline and Docker concepts and best practices.

Pipeline Introduction Exercise

To reinforce today's material we have an exercise for you.

Before our next session, please create a simple pipeline using the knowledge you obtained so far. Start by creating a new Pipeline job and explore the available steps through the Snippet Generator. Once you are familiar with the steps, create a simple pipeline from one of the projects you've been working with. Feel free to use the environment we set up earlier. Please note that only JDK and Git are installed. If you need some other tool (Maven, Gradle, NodeJS, and so on), you should first make sure that they are properly set up. As a alternative, you can do this exercise on your own Jenkins instance. In case you opt for this option, please note that the CloudBees Pipeline plugin needs to be installed. If, no the other hand, you are already using the CloudBees Jenkins Enterprise Edition, the plugin is already installed and you can start using it right away.

Tasks

- Create A New Pipeline Job
- Explore The Pipeline Syntax
- Implement A Simple Pipeline

Go back to slides

Docker Introduction Exercise

To reinforce today's material we have an exercise for you.

Before our next session, please create a simple pipeline using the knowledge you obtained so far. Start by creating a new Pipeline job and explore the available steps through the Snippet Generator. Once you are familiar with the steps, create a simple pipeline from one of the projects you've been working with. Feel free to use the environment we set up earlier. Please note that only JDK and Git are installed. If you need some other tool (Maven, Gradle, NodeJS, and so on), you should first make sure that they are properly set up. As a alternative, you can do this exercise on your own Jenkins instance. In case you opt for this option, please note that the CloudBees Pipeline plugin needs to be installed. If, no the other hand, you are already using the CloudBees Jenkins Enterprise Edition, the plugin is already installed and you can start using it right away.

The solution to all tasks is located at the end. Please try to solve them by yourself and look at the solutions only if you get stuck or want to validate your work.

The exercise consists of performing the whole delivery lifecycle of an application using only Docker. We'll test the application, and build and push a Docker container. The service we'll use is written in Go. Since this training is language-agnostic, you are not expected to know the language. You will not need any additional tools. All the tasks should be completed with Docker.

Before diving into the exercise, please SSH into the machine you are assigned and enter the */mnt/training-books-ms/exercises* directory. All the code you'll need is inside.

Task: Test the code

The command to test the code is as follows.

```
sh -c "go get -t && go test --cover -v"
```

Since Go is not installed on the server, you should use the golang image available in the Docker Hub.

The requirements for this task are as follows.

- Remove the container after the execution of the tests is finished
- The current host directory should be mounted as the **/go/src/docker-flow** volume.
- The **/go/src/docker-flow** inside the container should be the working directory.
- Run the above mentioned command

For additional information, please consult Docker Run documentation.

Task: Build the binary

The command to build the binary is as follows.

```
sh -c "go get && go build -v -o docker-flow-proxy"
```

The requirements for this task are as follows.

- Remove the container after the execution of the tests is finished

- The current host directory should be mounted as the **/go/src/docker-flow** volume.
- The **/go/src/docker-flow** inside the container should be working directory.
- Run the above mentioned command

For additional information, please consult Docker Run documentation.

Task: Create Dockerfile that defines the Docker image

The Dockerfile that we'll create has the following requirements.

- It should extend the `haproxy:1.6-alpine` image. Alpine is the smallest base image. Extending it will guarantee that our container is as small as it can be.
- Define maintainer's name and email address.
- It should download **Consul Template** binary (**https://releases.hashicorp.com/consul-template/0.13.0/consul-template_0.13.0_linux_amd64.zip**), place it into the **/usr/local/bin/** directory with the name **consul-template**, and make it executable. The following set of Linux commands would accomplish this requirement.

```
apk add --no-cache --virtual .build-deps curl unzip

curl -SL https://releases.hashicorp.com/consul-template/0.13.0/consul-
template_0.13.0_linux_amd64.zip -o /usr/local/bin/consul-template.zip

unzip /usr/local/bin/consul-template.zip -d /usr/local/bin/

rm -f /usr/local/bin/consul-template.zip

chmod +x /usr/local/bin/consul-template
```

- Create a soft link from **/lib/libc.musl-x86_64.so.1** to **/lib64/ld-linux-x86-64.so.2**. The following set of Linux commands would accomplish this requirement.

```
mkdir /lib64

ln -s /lib/libc.musl-x86_64.so.1 /lib64/ld-linux-x86-64.so.2
```

- Create the directory **/cfg/tmpl**.
- Copy **haproxy.cfg** to **/cfg/haproxy.cfg**.
- Copy **haproxy.tmpl** to **/cfg/tmpl/haproxy.tmpl**.
- Copy **docker-flow-proxy** to **/usr/local/bin/docker-flow-proxy**.

- Add **execute permissions to */usr/local/bin/docker-flow-proxy**.
- Define an environment variable **CONSUL_ADDRESS** with an empty string as value.
- Expose ports **80** and **8080**.
- Define `docker-flow-proxy server` as container's command.

For additional information, please consult Dockerfile reference documentation.

Task: Build the image and push it to the local registry

The requirements for this task are as follows.

- Build the image defined in the **Dockerfile** and name it **localhost:5000/docker-flow-proxy**
- Push the newly built image to the private registry running in **localhost:5000**.

For additional information, please consult Docker Build, Docker Tag, and Docker Push documentation.

Task: Run latest version of the container

The requirements for this task are as follows.

- List all the Docker processes.
- If the container with the same name is already running, remove it.
- Run the newly built container.
- The container should run in background.
- The name of the container should be **docker-flow-proxy**.
- Internal port **80** should be exposed to the host as **8081**.
- Internal port **8080** should be exposed to the host as **8082**.

For additional information, please consult Docker Run documentation.

Solution: Test the code

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

docker run --rm \
    -v $PWD:/go/src/docker-flow \
    -w /go/src/docker-flow \
    golang \
    sh -c "go get -t && go test --cover -v"
```

The explanation of the arguments is as follows.

- The `--rm` argument automatically removes the container when it exits
- The `-v $PWD:/go/src/docker-flow` argument mounts the current directory ($PWD) as `/go/src/docker-flow`.
- The `go get -t && go test --cover -v` argument is the command that is run inside the container

Solution: Build the binary

```
docker run --rm \
    -v $PWD:/go/src/docker-flow \
    -w /go/src/docker-flow \
    golang \
    sh -c "go get && go build -v -o docker-flow-proxy"
```

The explanation of the arguments is as follows.

- The `--rm` argument automatically removes the container when it exits
- The `-v $PWD:/go/src/docker-flow` argument mounts the current directory ($PWD) as `/go/src/docker-flow`.
- The `go get && go build -v -o docker-flow-proxy` argument is the command that is run inside the container

Solution: Create Dockerfile that defines the Docker image

The content of the **Dockerfile** should be as follows.

```
FROM haproxy:1.6-alpine
MAINTAINER  Viktor Farcic <vfarcic@cloudbees.com>

RUN apk add --no-cache --virtual .build-deps curl unzip && \
    curl -SL \
    https://releases.hashicorp.com/consul-template/0.13.0/consul-
template_0.13.0_linux_amd64.zip \
    -o /usr/local/bin/consul-template.zip && \
    unzip /usr/local/bin/consul-template.zip -d /usr/local/bin/ && \
    rm -f /usr/local/bin/consul-template.zip && \
    chmod +x /usr/local/bin/consul-template && \
    apk del .build-deps

RUN mkdir /lib64 && ln -s /lib/libc.musl-x86_64.so.1 /lib64/ld-linux-x86-64.so.2
RUN mkdir -p /cfg/tmpl
COPY haproxy.cfg /cfg/haproxy.cfg
COPY haproxy.tmpl /cfg/tmpl/haproxy.tmpl
COPY docker-flow-proxy /usr/local/bin/docker-flow-proxy
RUN chmod +x /usr/local/bin/docker-flow-proxy

ENV CONSUL_ADDRESS ""
EXPOSE 80
EXPOSE 8080

CMD ["docker-flow-proxy", "server"]
```

The explanation of the Dockerfile instructions is as follows.

- The `FROM` instruction sets the Base Image to **haproxy:1.6-alpine**.
- The `MAINTAINER` instruction allows you to set the Author field of the generated images.
- The `RUN` instruction executes any commands and commit the results.
- The `COPY` instruction copies new files or directories from the source (the first argument) and adds them to the filesystem of the container at the destination path (the second argument).
- The `ENV` instruction sets the environment variable (the first argument) to the value (the second argument).
- The `EXPOSE` instruction informs Docker that the container listens on the specified network ports at runtime.
- The `CMD` instruction provides defaults for an executing container.

Please use your favourite editor (e.g. **vi**) to create the **Dockerfile** or `echo` the contents that follow.

```
echo '
FROM haproxy:1.6-alpine
MAINTAINER  Viktor Farcic <vfarcic@cloudbees.com>

RUN apk add --no-cache --virtual .build-deps curl unzip && \
    curl -SL \
    https://releases.hashicorp.com/consul-template/0.13.0/consul-
template_0.13.0_linux_amd64.zip \
    -o /usr/local/bin/consul-template.zip && \
    unzip /usr/local/bin/consul-template.zip -d /usr/local/bin/ && \
    rm -f /usr/local/bin/consul-template.zip && \
    chmod +x /usr/local/bin/consul-template && \
    apk del .build-deps

RUN mkdir /lib64 && ln -s /lib/libc.musl-x86_64.so.1 /lib64/ld-linux-x86-64.so.2
RUN mkdir -p /cfg/tmpl
COPY haproxy.cfg /cfg/haproxy.cfg
COPY haproxy.tmpl /cfg/tmpl/haproxy.tmpl
COPY docker-flow-proxy /usr/local/bin/docker-flow-proxy
RUN chmod +x /usr/local/bin/docker-flow-proxy

ENV CONSUL_ADDRESS ""
EXPOSE 80
EXPOSE 8080

CMD ["docker-flow-proxy", "server"]
' | sudo tee Dockerfile
```

Solution: Build the image and push it to the local registry

```
docker build -t localhost:5000/docker-flow-proxy .

docker push localhost:5000/docker-flow-proxy
```

The `build` command builds Docker images from a Dockerfile and a context. The explanation of the `build` arguments is as follows.

- The `-t` argument specifies the name of the image. In this case, the name is prefixed with the IP and the port of the registry where the image will be pushed.
- The last argument is the path of the context. In this case, the `.` (dot) argument is translated to the current directory.

The `push` command is used to send images to the Docker Hub registry or to a self-hosted one. The explanation of the `tag` arguments is as follows.

- The first argument represents the image that will be pushed. It is prefixed with the IP and the port of the registry where the image will be pushed.

Solution: Run latest version of the container

```
docker ps -a

docker rm -f docker-flow-proxy

docker run -d \
    --name docker-flow-proxy \
    -p 8081:80 -p 8082:8080 \
    localhost:5000/docker-flow-proxy
```

The explanation of the `ps` arguments is as follows.

- The `-a` argument shows all containers (default shows just running)

The explanation of the `rm` arguments is as follows.

- The `-f` argument forces the removal of a running container.
- The `docker-flow-proxy` argument is the name of the container that should be removed.

The explanation of the `run` arguments is as follows.

- The `-d` argument runs the container in background.
- The `--name` argument assigns a name to the container.
- The `-p` argument publishes a container's port to the host. In this case, each value is split with colon (**:**). The left hand-side represents the port that should be published on the host while the right-hand side is the port defined internally inside the container.
- The last argument is the name of the image.

Go back to slides

The Project - Part 1: Exercise

The solution to all tasks is located at the end. Please try to solve them by yourself and look at the solutions only if you get stuck or want to validate your work.

The exercise consists of performing the whole delivery lifecycle of an application using Jenkins Pipeline and Docker. We'll test the application, and build and push a Docker container. The service we'll use is written in Go. Since this training is language-agnostic, you are not expected to know the language. You will not need any additional tools. All the tasks should be completed with Docker.

Before diving into the exercise, please SSH into the machine you are assigned and enter the */mnt/training-books-ms/exercises* directory. All the code you'll need is inside.

Task: Create a new Pipeline job

The requirements for this task are as follows.

- Create a new **Pipeline** job called **docker-flow-proxy**.

For additional information, please consult Getting Started tutorial.

Task: Clone the code from the GitHub repository

The requirements for this task are as follows.

- Everything should run inside the node called **cd**.
- Clone of **pipeline** branch of the **https://github.com/cloudbees/training-books-ms** repository.

For additional information, please consult node and git documentation.

Task: Run the tests and build the binary

The requirements for this task are as follows.

- Everything should run inside the node called **cd**.
- Define the stage called **test**.
- Run the tests and build the binary inside the **golang** container.
- Set user and group to **0**.
- Run the following commands inside the container.

```
ln -s $PWD /go/src/docker-flow
cd /go/src/docker-flow && go get -t && go test --cover -v
cd /go/src/docker-flow && go build -v -o docker-flow-proxy
```

For additional information, please consult withDockerContainer and stage documentation.

Task: Build the image and push it to the local registry

The requirements for this task are as follows.

- Everything should run inside the node called **cd**.
- Define the stage called **build**.
- Build the container called **localhost:5000/docker-flow-proxy**.
- Build the container called **localhost:5000/docker-flow-proxy**.

Task: Archive the binary

The requirements for this task are as follows.

- Everything should run inside the node called **cd**.
- Archive the binary **docker-flow-proxy**.

For additional information, please consult archive documentation.

Task: Run latest version of the container

The requirements for this task are as follows.

- Define the checkpoint called **deploy**
- Everything should run inside the node called **production**.
- Define the stage called **deploy**.
- Delete the container called **docker-flow-proxy**. Make sure that the pipeline does not fail if the container is not running.
- Run the **localhost:5000/docker-flow-proxy** container.
- The name of the container should be **docker-flow-proxy**.
- The internal port **80** should be exposed to the host as **8081**.
- The internal port **8080** should be exposed to the host as **8082**.

Solution: Create a new Pipeline job

- Open the Jenkins UI.
- Click the **New Item** link from the left-hand menu.
- Type **docker-flow-proxy** in the **Item Name** field, select the **Pipeline** job type, and click the **OK** button.

Solution: Clone the code from the GitHub repository

Write the following script inside the **Pipeline Script** field.

```
node("cd") {
    git branch: 'pipeline', url: 'https://github.com/cloudbees/training-books-ms'
}
```

The explanation of the snippet is as follows.

- The steps executed inside the `node` block will run in one of the slaves named or labeled **cd**.

Solution: Run the tests and build the binary

Add the following snippet below the `git` instruction inside the **Pipeline Script** field.

```
    stage 'test'
    docker.image("golang").inside('-u 0:0') {
        sh 'ln -s $PWD /go/src/docker-flow'
        sh 'cd /go/src/docker-flow && go get -t && go test --cover -v'
        sh 'cd /go/src/docker-flow && go build -v -o docker-flow-proxy'
    }
```

The explanation of the snippet is as follows.

- The `stage` instruction defines a group of steps.
- The `docker.image` instruction specifies the name of the image that will be used.
- The arguments set for the `inside` function will be passed as Docker `run` arguments.
- The steps executed inside the `docker.image.inside` block will run inside the specified container.

Solution: Build the image and push it to the local registry

Add the following snippet below the `docker.image("golang").inside('-u 0:0')` block inside the **Pipeline Script** field.

```
    stage 'build'
    docker.build('localhost:5000/docker-flow-proxy')
    docker.image('localhost:5000/docker-flow-proxy').push()
```

The explanation of the snippet is as follows.

- The `stage` instruction defines a group of steps.
- The `docker.build` instruction builds the image called `localhost:5000/docker-flow-proxy`. The name itself follows the **<REGISTRY>/<IMAGE_NAME>** format.
- The `docker.image` instruction returns **Container**. The `push` command is given to the container **localhost:5000/docker-flow-proxy**.

Solution: Archive the binary

Add the following snippet below the `docker.image('localhost:5000/docker-flow-proxy').push()` instruction inside the **Pipeline Script** field.

```
    archive 'docker-flow-proxy'
```

The explanation of the snippet is as follows.

- The `archive` instruction stores all the files that match the specified pattern.

Solution: Run latest version of the container

```
checkpoint 'deploy'

node('production') {
    stage 'deploy'
    try {
        sh 'docker rm -f docker-flow-proxy'
    } catch(e) { }
    docker.image('localhost:5000/docker-flow-proxy').run('--name docker-flow-
proxy -p 8081:80 -p 8082:8080')
}
```

The explanation of the snippet is as follows.

- In case of a system failure, the `checkpoint` instruction allows restart of the job from that point.
- The steps executed inside the `node` block will run in one of the slaves named or labeled **production**.
- The `sh` step is used to run the Docker `rm` command that removes the container. To prevent possible failure in case such a container does not exist, the step is wrapped inside an `try/catch` statement.

Solution: The complete Pipeline script

The complete script is listed below.

```
node("cd") {
    git branch: 'pipeline', url: 'https://github.com/cloudbees/training-books-ms'

    stage 'test'
    docker.image("golang").inside('-u 0:0') {
        sh 'ln -s $PWD /go/src/docker-flow'
        sh 'cd /go/src/docker-flow && go get -t && go test --cover -v'
        sh 'cd /go/src/docker-flow && go build -v -o docker-flow-proxy'
    }

    stage 'build'
    docker.build('localhost:5000/docker-flow-proxy')
    docker.image('localhost:5000/docker-flow-proxy').push()
    archive 'docker-flow-proxy'
}

checkpoint 'deploy'

node('production') {
    stage 'deploy'
    try {
        sh 'docker rm -f docker-flow-proxy'
    } catch(e) { }
    docker.image('localhost:5000/docker-flow-proxy').run('--name docker-flow-
proxy -p 8081:80 -p 8082:8080')
}
```

Go back to slides

The Project - Part 2: Exercise

The solution to all tasks is located at the end. Please try to solve them by yourself and look at the solutions only if you get stuck or want to validate your work.

The exercise consists of practicing with other job types. Since this training is language-agnostic, you are not expected to know the language. You will not need any additional tools. All the tasks should be completed with Docker.

Before diving into the exercise, please SSH into the machine you are assigned and enter the */mnt/training-books-ms/exercises* directory. All the code you'll need is inside.

Task: Convert the job into a template

The requirements for this task are as follows.

- Convert the *docker-flow-proxy* Pipeline job into a Job template called *docker-flow-proxy-template*.

- Convert repository name and container name into job properties.
- Create a new job called *docker-flow-proxy-from-template*. The job should be based on the *docker-flow-template*.

Task: Create a Multi-Branch Pipeline

The requirements for this task are as follows.

- Create a new job called *docker-flow-proxy-from-git*.
- The type of the job should be *Multibranch Pipeline*.
- The repository with Jenkins file should be *https://github.com/cloudbees/training-books-ms*.
- The *master* branch should be excluded.

Solution: Convert the job into a template

- Click the *New Item* link located in the left-hand menu in the home screen.
- Type *docker-flow-template* as *Item Name*, select *Job Template*, and click the *OK* button.
- Type *Docker Flow Template* in the *Display Name* field.
- Click the *Add* button in the *Attributes* section, type *repository* as ID and *Repository* as name.
- Click the *Add* button in the *Attributes* section, type *container* as ID and *Container* as name.
- Select *Groovy template for Pipeline* as the *Tranformer* type.
- Write the following script inside the *Pipeline Script* field.

```
node("cd") {
    git branch: 'pipeline', url: "https://github.com/${repository}"

    stage 'test'
    docker.image("golang").inside('-u 0:0') {
        sh 'ln -s $PWD /go/src/docker-flow'
        sh 'cd /go/src/docker-flow && go get -t && go test --cover -v'
        sh 'cd /go/src/docker-flow && go build -v -o docker-flow-proxy'
    }

    stage 'build'
    docker.build("localhost:5000/${container}")
    docker.image("localhost:5000/${container}").push()
    archive '${container}'
}

checkpoint 'deploy'

node('production') {
    stage 'deploy'
    try {
        sh "docker rm -f ${container}"
    } catch(e) { }
    docker.image("localhost:5000/${container}").run("--name ${container} -p
8081:80 -p 8082:8080")
}
```

- Click the *New Item* link located in the left-hand menu in the home screen.
- Type *docker-flow-proxy-from-template* as *Item Name*, select *Docker Flow Template*, and click the *OK* button.
- Set *docker-flow-proxy-from-template* as *Name, _cloudbees/training-books-ms* as *Repository*, and *docker-flow-proxy* as *Container*.
- Click the *Save* button.

Solution: Create a Multi-Branch Pipeline

- Click the *New Item* link located in the left-hand menu in the home screen.
- Type *docker-flow-proxy-from-git* as *Item Name*, select *Multibranch Pipeline*, and click the *OK* button.
- Select *Git* from the *Add Source* drop-down.
- Type *https://github.com/cloudbees/training-books-ms* as *Project Repository*.
- Click the *Advanced* button and type *master* in the *Exclude branches* field.
- Click the *Save* button.

Go back to slides