Continuous Delivery Pipeline

Raúl Estrada

Octubre 2020

Inventory

In their simplest form, we can have two environments: staging and production, each having one Docker host machine. In real life, we may want to add more host groups for each environment if we want to have servers in different locations or having different requirements.

Let's create two Ansible inventory files. Starting from the staging, we can define the `inventory/staging` file. Assuming the staging address is `192.168.0.241`, it would have the following content:

```
[webservers]
web1 ansible_host=192.168.0.241 ansible_user=admin
```

By analogy, if the production IP address is `192.168.0.242`, then the `inventory/production` should look like this:

```
[webservers]
web2 ansible_host=192.168.0.242 ansible_user=admin
```

It may look oversimplified to have just one machine for each environment; however, using Docker Swarm (which we show later in this book), a cluster of hosts can be hidden behind one Docker host.

Having the inventory defined, we can change acceptance testing to use the staging environment.

Acceptance testing environment

Depending on our needs, we could test the application by running it on the local Docker host (like we did in the previous chapter) or using the remote staging environment. The former solution is closer to what happens in production, so it can be considered as a better one. This is very close to what was presented in the *Method 1: Jenkins-first acceptance testing* section of the previous chapter. The only difference is that now we deploy the application on a remote Docker host.

In order to do this, we could use `docker` (or the `docker-compose` command) with the `-H` parameter, which specifies the remote Docker host address. This would be a good solution and if you don't plan to use Ansible or any other configuration management tool, then that is the way to go. Nevertheless, for the reasons already mentioned in this chapter, it is beneficial to use Ansible. In that case, we can use the `ansible-playbook` command inside the Continuous Delivery pipeline.

```
stage("Deploy to staging") {
    steps {
        sh "ansible-playbook playbook.yml -i inventory/staging"
    }
}
```

If `playbook.yml` and docker-compose.yml look the same as in the *Ansible with Docker* section, then it should be enough to deploy the application with dependencies into the staging environment.

Release

The production environment should be as close to the staging environment as possible. The Jenkins step for the release should also be very similar to the stage that deploys the application to the staging environment.

In the simplest scenario, the only differences are the inventory file and the application configuration (for example, in case of a Spring Boot application, we would set a different Spring profile, which results in taking a different properties file). In our case, there are no application properties, so the only difference is the inventory file.

```
stage("Release") {
    steps {
        sh "ansible-playbook playbook.yml -i inventory/production"
    }
}
```

Smoke testing

A smoke test is a very small subset of acceptance tests whose only purpose is to check that the release process is completed successfully. Otherwise, we could have a situation in which the application is perfectly fine; however, there is an issue in the release process, so we may end up with a non-working production.

The smoke test is usually defined in the same way as the acceptance test. So the `Smoke test` stage in the pipeline should look like this:

```
stage("Smoke test") {
    steps {
        sleep 60
        sh "./smoke_test.sh"
    }
}
```

After everything is set up, the Continuous Delivery build should run automatically and the application should be released to production. With this step, we have completed the Continuous Delivery pipeline in its simplest, but fully productive, form.

Complete Jenkinsfile

To sum up, throughout the recent chapters we have created quite a few stages, which results in a complete Continuous Delivery pipeline that could be successfully used in many projects.

Next we see the complete Jenkins file for the calculator project:

```
pipeline {
  agent any

  triggers {
    pollSCM('* * * * *')
  }

  stages {
    stage("Compile") { steps { sh "./gradlew compileJava" } }
    stage("Unit test") { steps { sh "./gradlew test" } }

    stage("Code coverage") { steps {
      sh "./gradlew jacocoTestReport"
      publishHTML (target: [
              reportDir: 'build/reports/jacoco/test/html',
              reportFiles: 'index.html',
              reportName: "JaCoCo Report" ])
      sh "./gradlew jacocoTestCoverageVerification"
    } }
```

```
stage("Static code analysis") { steps {
  sh "./gradlew checkstyleMain"
  publishHTML (target: [
          reportDir: 'build/reports/checkstyle/',

          reportFiles: 'main.html',
          reportName: "Checkstyle Report" ])
} }

stage("Build") { steps { sh "./gradlew build" } }


stage("Docker build") { steps {
  sh "docker build -t leszko/calculator:${BUILD_TIMESTAMP} ."
} }
```

```
    stage("Docker build") { steps {
      sh "docker build -t leszko/calculator:${BUILD_TIMESTAMP} ."
   } }

    stage("Docker push") { steps {
      sh "docker push leszko/calculator:${BUILD_TIMESTAMP}"
    } }

    stage("Deploy to staging") { steps {
      sh "ansible-playbook playbook.yml -i inventory/staging"
      sleep 60
    } }

    stage("Acceptance test") { steps { sh "./acceptance_test.sh" } }

    // Performance test stages

    stage("Release") { steps {
      sh "ansible-playbook playbook.yml -i inventory/production"
      sleep 60
    } }

    stage("Smoke test") { steps { sh "./smoke_test.sh" } }
  }
}
```