Week 5

Thus far, we have learned how Jenkins manages a "pipeline". The pipeline provides a way to define any automation process in a Jenkinsfile. In our case, the pipeline builds software and is a CI/CD environment.

This week we will add a testing stage to the pipeline. We will use Java and the "gradle" build system to construct and test software. The concepts and tools we shall look at all have analogies in other popular languages such as python, go, and C++.

Some ways to test code.

- Checkstyle - a way to ensure code is clean and readable. Code should not be messy. This way any developer will see a common format as opposed to an individual's style. The rules that define how code should look are in a specification, and the tool verifies code adheres to the rules.
- Static code analysis - looks for bugs in the code without running it. It will scan the code and report problems.
- Code coverage - checks the percentage of code paths exercised in the tests, gives an indication of testing quality and completeness.
- Unit tests - these are small, focused tests to check parts of the code for correctness
- Integration tests - these are larger tests to ensure the overall system is working. We will study those later in the course as we build out a system in kubernetes.

These tools are often capable of producing reports in HTML which can then be displayed in Jenkins. The reports can show the number and severity of problems, their location in the code, and other facts.

The data these tools generate can show trends. For example, as new unit tests are added, code coverage should increase.

A devOps team should closely monitor the "health" of the software, and quickly fix problems as they arise. The team should strive to continuously improve the software's health.

Lab 1

In this lab, we will create a pipeline that

1. Creates an agent that runs a docker image containing the "gradle" build environment
2. Downloads source code from github
3. Builds the code using gradle
4. Runs unit tests
5. Runs code coverage tests
6. Generates a report
7. Displays the report in Jenkins

We will figure out how to create a pipeline by first running the commands manually in a stand-alone pod.

Create the pod and get into its shell this way:

```
kubectl run gradle --rm -it  --image=gradle:6.3-jdk14 -- /bin/bash
```

Once inside, clone this week's homework.

```
git clone
https://github.com/dlambrig/Continuous-Delivery-with-Docker-and-Jenkins-Second-Edition
```

...and have a look around. We will be working on Chapter08/sample1. When Jenkins runs, it will create a pod with this image and compile code in this directory.

```
cd Continuous-Delivery-with-Docker-and-Jenkins-Second-Edition/Chapter08/sample1/
```

You run gradle using the ./gradlew executable (run the chmod command). It is configured in file build.gradle.

```
chmod +x gradlew
```

To see what gradle is configured to do, and check its status:

```
./gradlew tasks
./gradlew --status
```

Lots of test options are available. Unit tests:

```
./gradlew test
```

Try code coverage:

```
./gradlew jacocoTestCoverageVerification
```

This one fails. You will have to fix it in the homework assignment.

Checkstyle is done as follows. Its reports are put in directory: build/reports/checkstyle/main.html

```
./gradlew checkstyleMain
```

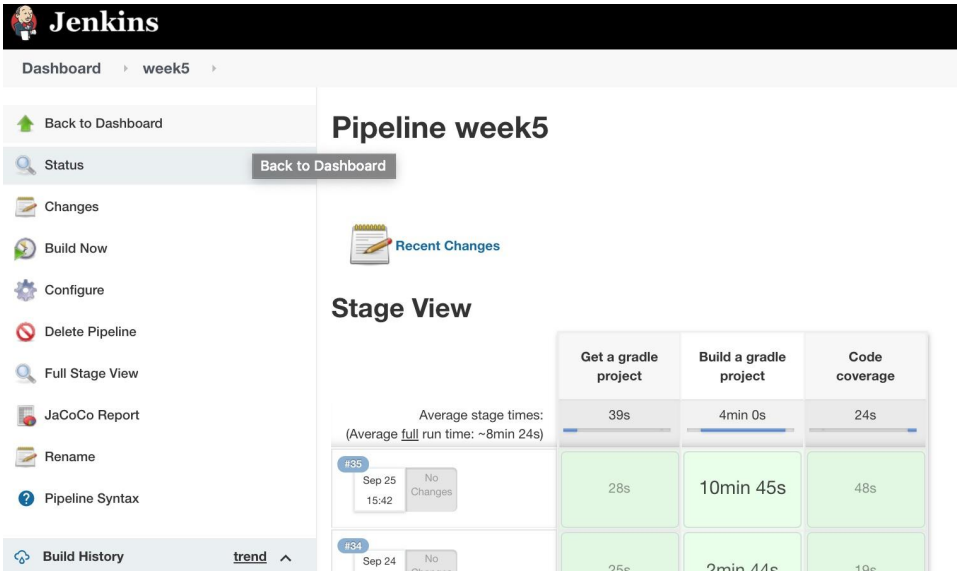
Now, create a pipeline to run those commands (see next page).

You should first **download the Jenkins plugin** "HTML Publisher plugin".

Notice a new section in the Jenkinsfile. See also "jacocoTestReport":

```
                    publishHTML (target: [
                      reportDir: 'Chapter08/sample1/build/reports/jacoco/test/html',
                      reportFiles: 'index.html',
                      reportName: "JaCoCo Report"
                    ])
.
.
./gradlew jacocoTestReport
```

JacocoTestReport produces output in the build/reports/jacoco/test/html/ subdirectory. The pipeline command above pulls file index.html out and gives it a name "JaCoCo Report". The "publishHTML" plugin will create a new button after the pipeline runs. Clicking on it shows the output.



Another debugging technique: set the pod retention to "always". This will keep the pod running rather than deleting it after the pipeline finishes it. You can then open a bash shell on it.

```
kubectl exec -it <pod name> -n devops-tools — /bin/bash
```

Once on , you can debug the commands the pipeline will run in the container.

The following example uses "scripted" syntax in Jenkins. The newer "declarative" syntax can also be used and will be used in coming weeks in the class.

In the script, a pod template and container template are defined. These define everything needed to run a pod, in a manner similar to yaml files. By default, commands will run in the jnlp container defined in last week's configuration, but in this week's example, a second container is defined to run "gradle", which is this week's Java code building tool. So, two containers will be created in the agent (pod).

The git plugin and the publishHTML plugin are also used in the example. The git plugin provides a command to download the repository. The publishHTML plugin generates a summary of results viewable on the web.

```
// week5 example uses Jenkin's "scripted" syntax, as opposed to its "declarative" syntax
// see: https://www.jenkins.io/doc/book/pipeline/syntax/#scripted-pipeline

// Defines a Kubernetes pod template that can be used to create nodes.

podTemplate(containers: [
    containerTemplate(
        name: 'gradle', image: 'gradle:6.3-jdk14', command: 'sleep', args: '30d'
        ),
    ]) {

    node(POD_LABEL) {
        stage('Run pipeline against a gradle project') {
             // "container" Selects a container of the agent pod so that all shell steps are
            // executed in that container.
            container('gradle') {
                stage('Build a gradle project') {
                    // from the git plugin
                    // https://www.jenkins.io/doc/pipeline/steps/git/
                    git
'https://github.com/dlambrig/Continuous-Delivery-with-Docker-and-Jenkins-Second-Edition.git'
                    sh '''
                    cd Chapter08/sample1
                    chmod +x gradlew
                    ./gradlew test
                    '''
                }

                stage("Code coverage") {
                        try {
                        sh '''
                           pwd
                                  cd Chapter08/sample1
                             ./gradlew jacocoTestCoverageVerification
                              ./gradlew jacocoTestReport
                        '''
                    } catch (Exception E) {
                        echo 'Failure detected'
                    }

                    // from the HTML publisher plugin
                    // https://www.jenkins.io/doc/pipeline/steps/htmlpublisher/
                    publishHTML (target: [
                      reportDir: 'Chapter08/sample1/build/reports/tests/test',
                       reportFiles: 'index.html',
                       reportName: "JaCoCo Report"
                    ])
                }
            }
        }
    }
}
```