Continuous Delivery with Jenkins Pipeline

Lev Epshtein

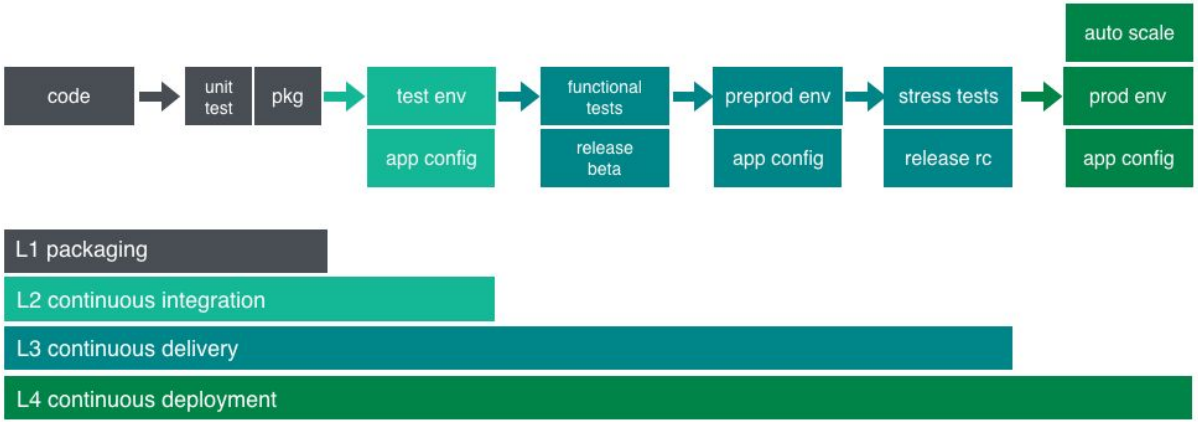
What is Jenkins

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed

The Continuous Delivery Pipeline

Enables a constant flow of changes into production by means of an automated software production line, by breaking down the software delivery process into stages.



https://dzone.com/articles/what-is-continuous-delivery-pipeline

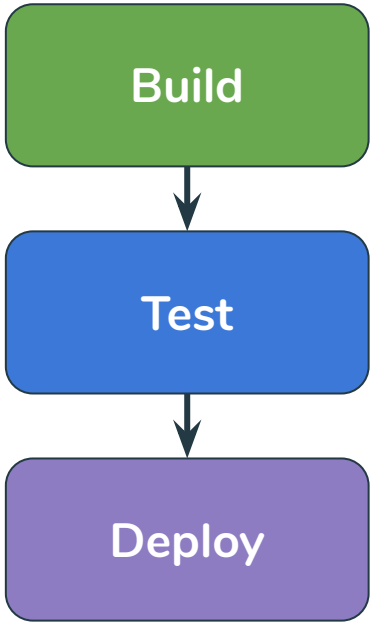
Introduction to Jenkins Pipeline

- Historically, the default interaction model with Jenkins has been using web UI, requiring users to create jobs and fill in the details manually through a web browser
- Disadvantages:
  - Huge effort to create and manage jobs.
  - CI configuration separated from the code.

Pipeline as Code

- After the introduction of the Pipeline plugin, now it’s possible to implement an entire delivery/deployment pipeline in a Jenkinsfile and store that alongside the source control
- Advantages:
  - Checking the pipeline definition into source control.
  - Support for extending the DSL by means of the shared libraries feature.

Pipeline as Code - Example




```
pipeline {
    agent any

    stages {
        stage('Build'){
            steps {
                sh 'make'
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml'
            }
        }
        stage('Deploy'){
            steps {
                sh 'make publish'
            }
        }
    }
}
```

Scripted vs Declarative

- Jenkins pipelines are based on groovy programming languages
- Scripted pipeline versus declarative pipeline
  - Syntax and flexibility
- Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write. This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.
- Whereas, the scripted pipeline is a traditional way of writing the code. In this pipeline, the Jenkinsfile is written on the Jenkins UI instance.
- The declarative pipeline is defined within a block labelled **'pipeline'** whereas the scripted pipeline is defined within a **'node'**.

Scripted Pipeline
Learn by Demos

<>/nice-devops-supplementary/Jenkins/pipeline-example/first-demo-scripted-pipeline
<>/nice-devops-supplementary/Jenkins/pipeline-example/second-demo-scripted-steps_pipeline

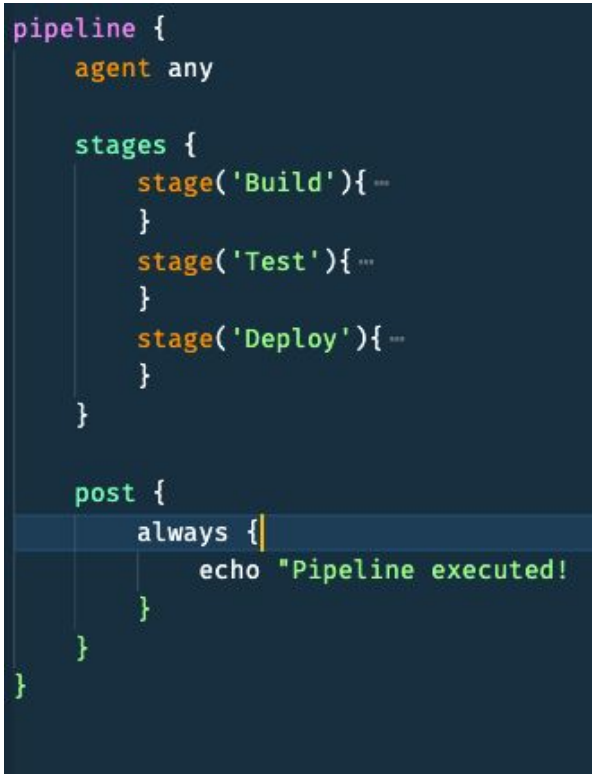
Declarative Pipeline

Declarative Pipeline

- Simplified and opinionated syntax on top of the pipeline sub-systems
- Follows the groovy syntax
- The top-level of the pipeline must be a pipeline { .. } block
- Inside of pipeline the blocks consist of sections, directives, steps, or assignment statements.

Sections

- Usually contains directive or steps
- Common sections:
  - **Agent** - defines on which agent the pipeline or the stage will be executed
  - **Stage** - define the stages of the pipeline
  - **Post** - define the post build steps



Directives

- Can stay inside of a section or pipeline block
- Common directives:
  - **Triggers** - automatically triggers the pipeline based on definition.
  - **Parameters** - define one or more parameters which an user should provide when triggering the pipeline
  - **Tools** - define the tools configured on Jenkins
  - **Stage** - inside of stage section; contains steps and agent
  - **When** - gives control to what should be executed in the pipeline

```
pipeline {
    agent any

    tools {
        maven 'M3'
    }

    parameters {
        string(name: 'VERSION',
               defaultValue: '1.0.0',
               description: 'What is the version to build?')
    }

    stages {
        stage('Build'){
            steps {
                sh "./build.sh ${params.VERSION}"
            }
        }
    }
}
```

Steps

Defines all the steps of given stage

```
pipeline {
    agent any

    stages {

        stage('Build'){

            steps {
                sh "./build.sh ${params.VERSION}"
                echo "Print a message after the build"
            }
        }
    }
}
```

Script Inside of a Declarative Pipeline

Script is a step to use scripted pipeline inside of a declarative pipeline

```
pipeline {
    agent any

    stages {

        stage('Build'){

            steps {
                echo "Hello World"

                    script {
                        for (int = 0; i < 5; ++i) {
                            echo "Printing numbers ${i}"
                        }
                    }
            }
        }
    }
}
```