## agent

- The agent section specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment depending on where the agent section is placed
- The section can be defined at the top-level inside the pipeline block or at stage-level

**Parameters:**

- **Any:** Execute the Pipeline, or stage, on any available agent
- **None:** When applied at the top-level of the pipeline block no global agent will be allocated for the entire Pipeline run and each stage section will need to contain its own agent section
- **Label:** Execute the Pipeline, or stage, on an agent available in the Jenkins environment with the provided label
- **Node:** behaves the same as agent { label 'labelName' }, but node allows for additional options

- **Run pipeline stage on group of machines using label**

```
pipeline {
    agent any
    stages {
        stage('Stage1') {
            steps {
                echo 'First Stage'
            }
        }
    }
}
```

- **Run 2 stages on group of machines**

```
pipeline {
    agent { label 'test' }
    stages {
        stage('Stage1') {
            steps {
                echo 'First Stage'
            }
        }
        stage('Stage2') {
            steps {
                echo 'Second Stage'
            }
        }
    }
}
```

- **Run Stage level agent section**

```
pipeline {
    agent none
    stages {
        stage('Stage1') {
            agent { label 'test' }
            steps {
                echo 'First Stage'
            }
        }
        stage('Stage2') {
            agent any
            steps {
                echo 'Second Stage'
            }
        }
    }
}
```

- **Custom Workspace for agent**

```
pipeline {
    agent none
    stages {
        stage('Stage1') {
            agent {
                node {
                    label 'test'          // comment : use \\ for windows dir separator
                    customWorkspace 'D:\\CLASSROOM\\JENKINS\\workspace2'
                }
            }
            steps {
                echo 'First Stage'
            }
        }
        stage('Stage2') {
            agent any
            steps {
                echo 'Second Stage'
            }
        }
    }
}
```

## Directives: environment

- **An environment directive used in the top-level pipeline block will apply to all steps within the Pipeline**

```
pipeline {
    agent { label 'test' }
    environment {
        MYNAME = 'Adam'
    }
    stages {
        stage('Stage1') {
            steps {
                bat "echo 'Your name: %MYNAME%'"
            }
        }
        stage('Stage2') {
            steps {
                echo env.MYNAME
            }
        }
    }
}
```

- **An environment directive defined within a stage will only apply the given environment variables to steps within the stage**

```
pipeline {
    agent { label 'test' }
    environment {
        VARVAL = 'global'
    }
    stages {
        stage('Stage1') {
            environment {
                VARVAL = 'local'
            }
            steps {
                bat "echo 'Your name: %VARVAL%'"
            }
        }
        stage('Stage2') {
            steps {
                echo env.VARVAL
            }
        }
```

## Directives: parameters

- Provides a list of parameters which a user should provide when triggering the Pipeline. The values for these user-specified parameters are made available to Pipeline steps via the 'params' object

**Available Parameters**

- String: A parameter of a string type

```
parameters { string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?') }
```

- Text: A text parameter, which can contain multiple lines

```
parameters {text(name: 'BIOGRAPHY', defaultValue: '''', description: 'Enter info') }
```

- booleanParam: A boolean parameter

```
parameters {booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')}
```

- Choice: A choice parameter

```
parameters {choice(name: 'CHOICE', choices: ['One', 'Two', 'Three'], description: 'Pick something')}
```

- File: A file parameter, which specifies a file to be submitted by the user when scheduling a build

```
parameters {file(name: "FILE", description: "Choose a file to upload")}
```

```
pipeline {
  agent any
  parameters {
    string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')

    text(name: 'BIOGRAPHY', defaultValue: '', description: 'Enter some information about the person')

    booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')

    choice(name: 'CHOICE', choices: ['One', 'Two', 'Three'], description: 'Pick something')

    password(name: 'PASSWORD', defaultValue: 'SECRET', description: 'Enter a password')

    file(name: "FILE", description: "Choose a file to upload")
  }
  stages {
    stage('Example') {
      steps {
        echo "Hello ${params.PERSON}"

        echo "Biography: ${params.BIOGRAPHY}"

        echo "Toggle: ${params.TOGGLE}"

        echo "Choice: ${params.CHOICE}"

        echo "Password: ${params.PASSWORD}"
      }
    }
  }
}
```

## Directives: options

- The options directive allows configuring Pipeline-specific options from within the Pipeline itself
- Pipeline provides a number of these options:

➢ buildDiscarder - Persist artifacts and console output for the specific number of recent Pipeline runs

```
pipeline {
  agent { label 'test' }
  options {
            buildDiscarder(logRotator(numToKeepStr: '5'))
  }
  stages {
    stage('Stage1') {
      steps {
        echo 'First Stage'
      }
    }
  }
}
```

➢ retry - On failure, retry the entire Pipeline the specified number of times

```
pipeline {
  agent { label 'test' }
  options {
        retry(3)
  }
  stages {
    stage('Stage1') {
      steps {
        bat 'waitfor test /t 10'
      }
    }
  }
}
```

➢ timeout - Set a timeout period for the Pipeline run, after which Jenkins should abort the Pipeline

```
pipeline {
  agent { label 'test' }
  options {
     timeout(time: 5, unit: 'SECONDS')
  }
  stages {
    stage('Stage1') {
      steps {
        bat 'waitfor test /t 25'
      }
    }
  }
}
```

➢ timestamps - Prepend all console output generated by the Pipeline run with the time at which the line was emitted

```
pipeline {
  agent { label 'test' }
  options {
            timeout(time: 5, unit: 'SECONDS')
            timestamps()
  }
  stages {
    stage('Stage1') {
      steps {
        bat 'waitfor test /t 25'
      }
    }
  }
}
```

➢ disableConcurrentBuilds - By default, we will be able to trigger multiple builds i.e concurrent executions of the pipeline, we can disallow concurrent executions of the Pipeline. Can be useful for preventing simultaneous accesses to shared resources

```
pipeline {
  agent { label 'test' }
  options {
            timestamps()
            disableConcurrentBuilds()
  }
  stages {
    stage('Stage1') {
      steps {
        bat 'waitfor test /t 25'
      }
    }
  }
}
```

# Directives: triggers

- The triggers directive defines the automated ways in which the Pipeline should be re-triggered.
- The triggers currently available are cron, pollSCM and upstream.

➢ cron - Persist artifacts and console output for the specific number of recent Pipeline runs

```
pipeline {
  agent { label 'test' }
  triggers {
    cron('* * * * *')
  }
  stages {
    stage('Stage1') {
      steps {
        echo 'test'
      }
    }
  }
}
```

➢ upstream - Accepts a comma separated string of jobs and a threshold.

➢ When any job in the string finishes with the minimum threshold, the Pipeline will be re-triggered

```
pipeline {
  agent { label 'test' }
  triggers {
    upstream(upstreamProjects: 'job1', threshold: hudson.model.Result.SUCCESS)
  }
  stages {
    stage('Stage1') {
      steps {
        echo 'test'
      }
    }
  }
}
```

# Directives: Steps

- Basically, steps tell Jenkins what to do and serve as the basic building block
- The steps section defines a series of one or more steps to be executed in a given stage directive

➢ git - It performs a clone from the specified repository, shorthand for the generic SCM step

```
pipeline {
  agent { label 'test' }
  stages {
    stage('Clone Repo') {
      steps {
        echo 'Going to Checkout from Git'
        git branch: 'master', url: 'https://github.com/scmlearningcentre/maven.git'
        echo 'Completed Checkout from Git'
      }
    }
  }
}
```

➢ build - Triggers a new build for a given **downstream** job & **waits for its completion**

```
pipeline {
  agent { label 'test' }
  stages {
    stage('Nightly') {
      steps {
        build 'Nightlyjob'
      }
    }
    stage('Stage2') {
      steps {
        echo 'Testing'
      }
    }
  }
}
```

➢ build - Triggers a new build for a given **downstream** job & **do not wait**

```
pipeline {
  agent { label 'test' }
  stages {
    stage('Build') {
      steps {
        build job: 'Nightlyjob', wait: false
      }
    }
    stage('Stage2') {
      steps {
        echo 'Testing'
      }
    }
  }
}
```

➢ mail - Sending email

```
pipeline {
  agent { label 'test' }
  stages {
    stage('mail') {
      steps {
        mail bcc: '', body: 'Hi Adam', cc: '', from: '', replyTo: '', subject: 'Test Mail', to: 'scmlearningcentre@gmail.com'
      }
    }
  }
}
```

- dir - Change current directory.
- Any step inside the dir block will use this directory as current and any relative path will use it as base path

```
pipeline {
  agent { label 'test' }
  stages {
    stage('Stage1') {
      steps {
          bat 'mkdir testfirst'
          dir('D:\\CLASSROOM') {
             bat 'mkdir test'
          }
          bat 'mkdir testlast'
      }
    }
  }
}
```

# Directives: When

- The when directive allows the Pipeline to determine whether the stage should be executed depending on the given condition. The when directive must contain at least one condition

- **Environment : Execute the stage when the specified environment variable is set to the given value**

```
pipeline {
    agent any
    environment { DEPLOY_TO = 'qa'}
    stages {
        stage('Stage1') {
            when {
                    environment name: 'DEPLOY_TO', value: 'qa'
                }
            steps {
                  echo 'Running Stage1 for QA'
            }
        }
       stage('Stage2') {
            when {
                        environment name: 'DEPLOY_TO', value: 'production'
                }
            steps {
                  echo 'Running Stage1 for production'
            }
        }
    }
}
```

- **expression : Execute the stage when the specified Groovy expression evaluates to true, Note that when returning strings from your expressions they must be converted to booleans or return null to evaluate to false. Simply returning "0" or "false" will still evaluate to "true"**

```
pipeline {
  agent any
          parameters {
    booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')
  }
  stages {
    stage('Stage1') {
      when {
                                    expression { return params.TOGGLE }
                                  }
      steps {
         echo 'Testing'
      }
    }
  }
}
```

- **equals : Execute the stage when the expected value is equal to the actual value**

```
pipeline {
  agent any
          parameters {
    string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')
  }
  stages {
    stage('Stage1') {
      when { equals expected: 'adam' , actual: params.PERSON }
      steps {
         echo 'Hi Adam !!'
      }
    }
  }
}
```

- **Not equals : Execute the stage when the nested condition is false. Must contain one condition**

```
pipeline {
  agent any
          parameters {
    string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')
  }
  stages {
    stage('Stage1') {
      when { not { equals expected: 'adam' , actual: params.PERSON }}
      steps {
         echo 'Hi Adam !!'
      }
    }
  }
}
```

- **allOf : Execute the stage when all of the nested conditions are true. Must contain at least one condition**

```
AND (allOf): Cond1 AND Cond2
-----------
True AND True = True
True AND False = False
False AND True = False
False AND False = False
```

```
pipeline {
    agent any
    parameters {
        string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')
        booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')
    }
    stages {
        stage('Stage1') {
            when {
                    allOf {
                            equals expected: 'adam', actual: params.PERSON
                            expression { return params.TOGGLE }
                        }
                    }
            steps {
                  echo 'Hi Adam !!'
            }
        }
    }
}
```

- anyOf : Execute the stage when at least one of the nested conditions is true. Must contain at least one condition

```
OR  (anyOf):Cond1 OR Cond2
True OR True = True
True Or False = True
False Or True = True
False Or False = False
```

```
pipeline {
    agent any
    parameters {
        string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')
        booleanParam(name: 'TOGGLE', defaultValue: true, description: 'Toggle this value')
    }
    stages {
        stage('Stage1') {
            when {
                    allOf {
                            equals expected: 'adam' , actual: params.PERSON
                            expression { return params.TOGGLE }
                        }
                }
            steps {
                  echo 'Hi Adam !!'
            }
        }
    }
}
```

- beforeagent : By default, the when condition for a stage will be evaluated after entering the agent for that stage, if one is defined.
- If beforeAgent is set to true, the when condition will be evaluated first, and the agent will only be entered if the when condition evaluates to true

```
pipeline {
    agent none
        parameters {
        string(name: 'PERSON', defaultValue: 'Mr Adam', description: 'Who are you?')
    }
    stages {
        stage('Stage1') {
            agent { label 'demo' }
            when {
                    beforeAgent true
                    equals expected: 'adam' , actual: params.PERSON
                }
            steps {
                  echo 'Hi Adam !!'
            }
        }
    }
}
```

## Parallel Stages

- Stages in Declarative Pipeline by default run in sequential order
- Stages in Declarative Pipeline may declare a number of nested stages within a parallel block
- Note that a stage must have one and only one of steps, stages, or parallel
- The nested stages cannot contain further parallel stages themselves
- Any stage containing parallel cannot contain agent
- Example: ** Create Empty dir D:\\CLASSROOM\\JENKINS\\para1 and D:\\CLASSROOM\\JENKINS\\para2 & current workspace

```
pipeline{
 stages{
   stage(stage1) {
   }
   stage(stage2) {
    parallel{
        stage(stage2.1) {
        }
        stage(stage2.2) {
        }
    }
   }
   stage(stage3) {
   }
 }
}
```

```
pipeline {
  agent any
  stages {
    stage('Stage 1') {
      steps {
          sh 'sleep 10'
      }
    }
    stage('Stage 2') {
     steps {
          sh 'sleep 10'
       }
    }
    stage('Stage 3') {
     parallel {
      stage('Parallel 1') {
        steps {
          sh 'sleep 10'
        }
      }
      stage('Parallel 2') {
        steps {
          sh 'sleep 10'
        }
      }
     }
    }
```

## Post

- The post section defines one or more additional steps that are run upon the completion of a Pipeline's or stage's run (depending on the location of the post section within the Pipeline)
- Post can support any of the following post-condition blocks:
  - always: Steps are executed regardless of the completion status.
  - changed: Executes only if the completion results in a different status than the previous run.
  - fixed: Executes only if the completion is successful and the previous run failed
  - regression: Executes only if current execution fails, aborts or is unstable and the previous run was successful.
  - aborted: Steps are executed only if the pipeline or stage is aborted.
  - failure: Steps are executed only if the pipeline or stage fails.
  - success: Steps are executed only if the pipeline or stage succeeds.
  - unstable: Steps are executed only if the pipeline or stage is unstable.

```
pipeline {
    agent any
    stages {
        stage('Example') {
            steps { echo 'Hello Students' }
        }
    }
    post {
        always {
            echo 'Hello again!'
        }
    }
}
```

```
pipeline {
    agent any
    stages {
        stage('Stage1') {
            steps { echo 'Stage 1' }
            post {
                always { echo 'Hello again!' }
            }
        }
        stage('Stage2') {
            steps { echo 'Stage 2' }
        }
    }
}
```

- **Handling failure:**

```
pipeline {
  agent any
  stages {
    stage('Example') {
      steps { bat "exit 1" }
    }
  }
  post {
      failure {
mail body: 'Hi Adam', subject: 'The Pipeline failed', to: 'scmlearningcentre@gmail.com'
    }
  }
}
```

**STAGE 0 (Triggering CI through Webhook)**

- Install "Gitlab Plugin" on Jenkins & restart Jenkins (https://plugins.jenkins.io/gitlab-plugin)
- Generate Personal Access Token in Gitlab i.e User settings -> Access tokens
- In Jenkins add credentials for Gitlab API token with the above value
- Once the API Access has been setup, we can configure the connection between Jenkins and GitLab
  - This happens in the Mange Jenkins -> Configure System -> Gitlab
  - Uncheck the box "Enable authentication for '/project' end-point"
- Create Webhook:
  - Go to the Gitlab project -> Settings -> Integrations
  - Add URL "http:<JenkinsURL>:8080/project/<JobName>"
  - Enable "Push Events" under Trigger
  - Uncheck "Enable SSL verification"

**STAGE 1 (Checkout Code)**

- Stage (Pre-Check): Evaluate whether to build the code or not, depending on the files modified in the commit

** REFER CLASS NOTES FOR DEMO SOURCECODE **

**STAGE 2 (Build Artifacts)**

** REFER CLASS NOTES FOR DEMO SOURCECODE **