endava

CONTINUOUS INTEGRATION WITH JENKINS

MARILENA ISTRATE

AGENDA

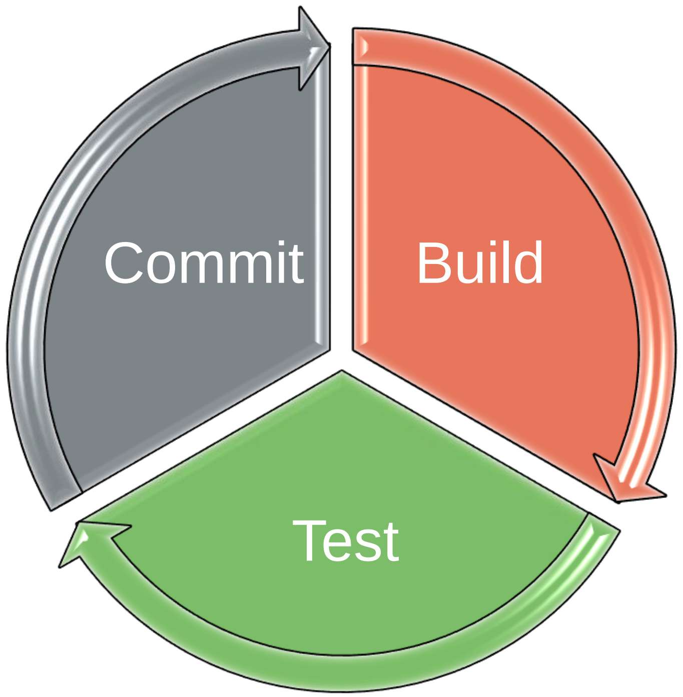
INTRODUCTION TO CI/CD

Continuous Integration is a development practice that requires developers to integrate code into a shared repository frequently, preferably several times a day.

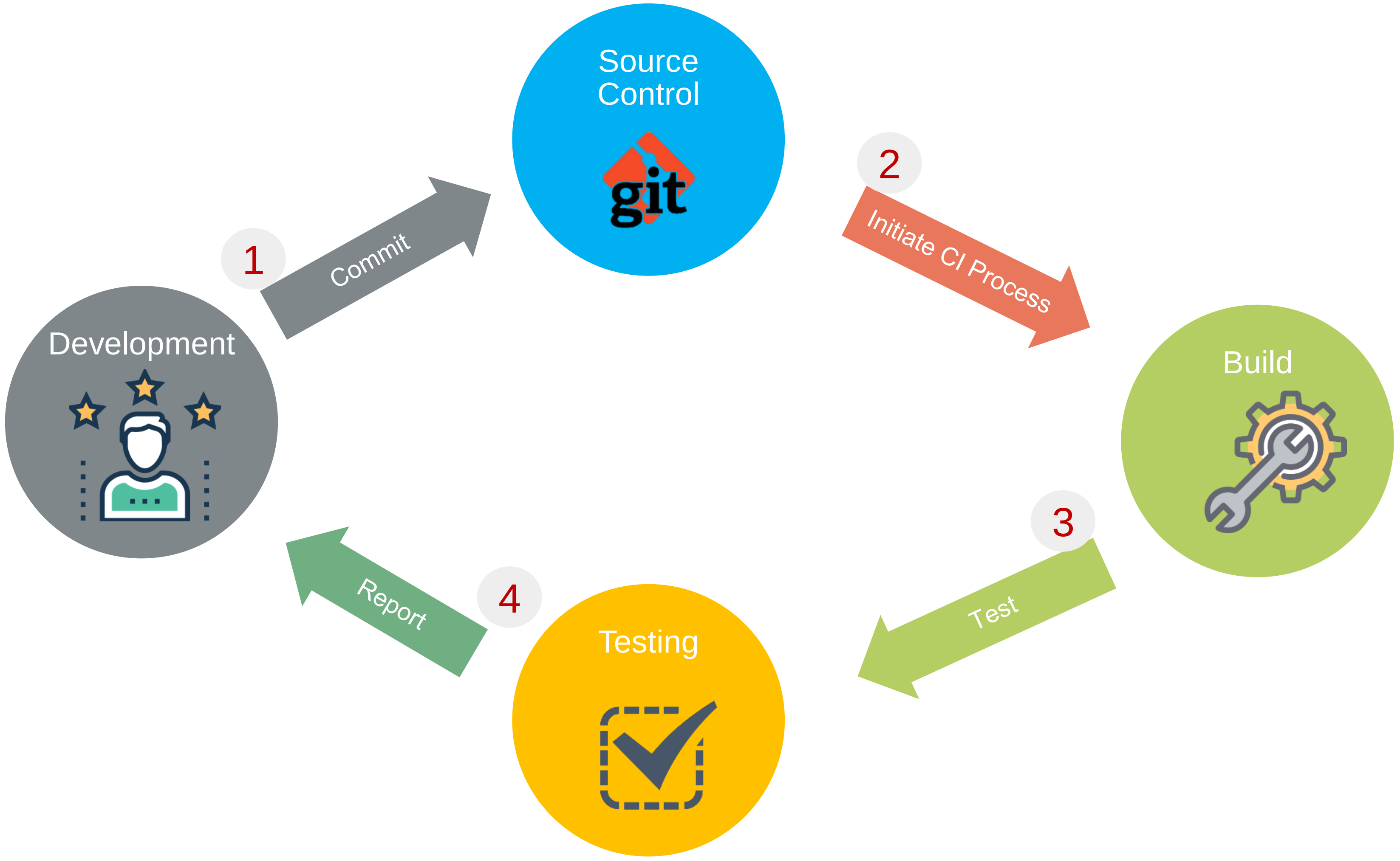
Committing code triggers an automated build system that will grab the latest code from the shared repository and will **build, test, and validate the entire application.**

Goal: to keep the software in running state as much as possible and decrease time needed for merging changes from hours and days to minutes

With continuous integration, your software is proven to work with every new change!



CONTINUOUS INTEGRATION



CONTINUOUS INTEGRATION

BENEFITS OF CI

- Faster product time to market
- Encourages the team to push every change without being afraid of breaking anything
- Reduces risk; it helps detect bugs and code defects earlier
- Constant feedback

TOOLS

- **Jenkins -** free, cross platform, server based
- **Travis CI -** Hosted on GitHub, free for open source project
- **Bamboo**
- **Team City -** suited for .NET projects, free and paid, server based

THE DIFFERENCE BETWEEN CI AND CD

CONTINUOUS INTEGRATION (CI)

Integrate your software early and often.

Fast and automated feedback on the correctness of your application every time there is a code change.

CONTINUOUS DELIVERY (CD)

Your software is always in a deployable state.

Fast, automated feedback on the production readiness of your application every time there is a change – to code, infrastructure or configuration.

WHAT DOES CD STAND FOR?

DELIVERY OR DEPLOYMENT?

Continuous Delivery ≠ Continuous Deployment

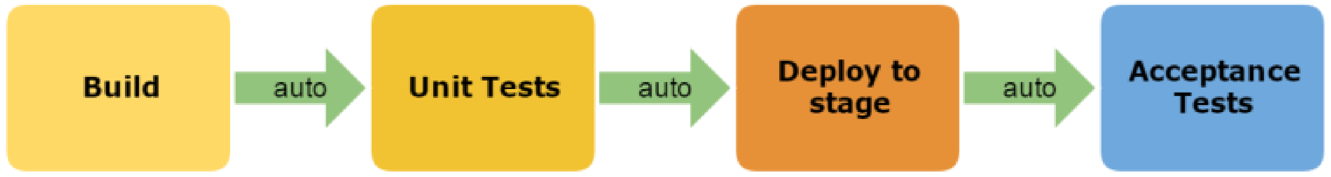
CONTINUOUS DELIVERY (CD)

Software is always deployable.

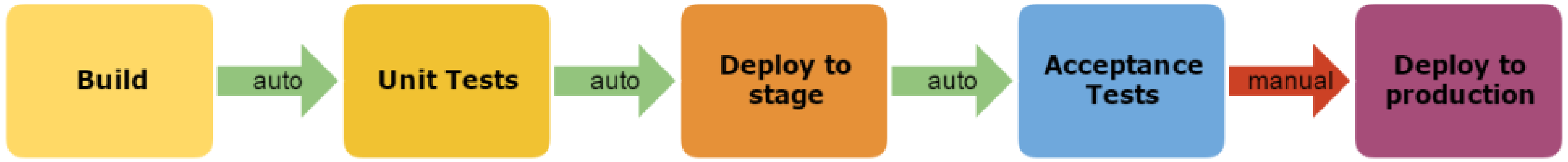
CONTINUOUS DEPLOYMENT (CD)

Software is deployed automatically, as the final stage of your deployment pipeline.

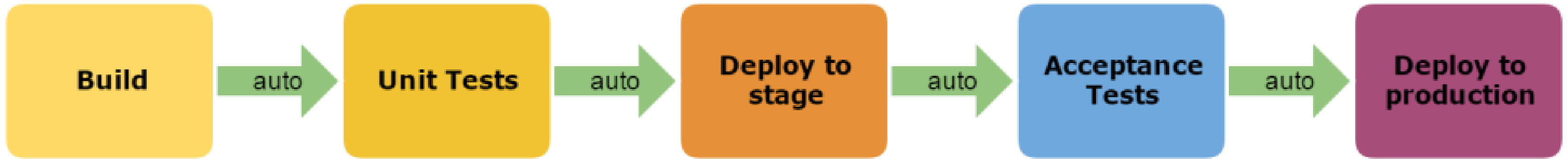
Continuous Integration



Continuous Delivery



Continuous Deployment



INTRODUCTION TO CI/CD

Who practices CD?

amazon Spotify NETFLIX

flickr endava facebook

Google Microsoft

INTRODUCTION TO CI/CD

HOW TO IMPLEMENT IT?

- Identify all manual steps for building / packaging your application from source code, as well as the instructions for configuring and successfully deploying it to production.
- Start automating your delivery pipeline incrementally.
- Fail faster and cheaper
- Increase code coverage by writing meaningful tests. Their main purpose is to fail the build for code that does not meet functional requirements.
- Use tools such as Jenkins to write your pipeline as code and store it in the same VCS repository with the rest of your application source code.
- Configure VCS server to trigger builds automatically on each change to your main branch.
- Define and implement ways for providing relevant feedback via emails, Slack notifications etc.
- Continually seek to improve and optimize the process.

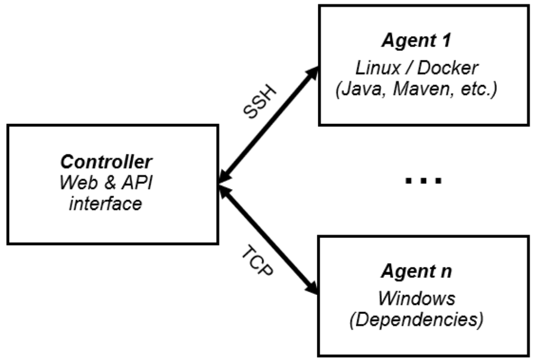
JENKINS

- 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
- Written in Java, forked from **Hudson** and maintained by **CloudBees (CloudBees CI** is a commercial extension of open source Jenkins)
- **Platform Independent**. Jenkins is available for all platforms and different operating systems, whether OS X, Windows or Linux.
- Available in 3 flavors: weekly releases, **Long Term Support (LTS)** releases and paid Enterprise
- No database, stores everything in XML files
- Many **plugins** available to bring a lot of useful functionalities
- **Web interface** for configuration and standard use

DISTRIBUTED BUILDS

JENKINS ARCHITECTURE IS FUNDAMENTALLY „CONTROLLER+AGENT"

- **Controller** machine (previously called Master) is designed to do co-ordination and provide the GUI and API endpoints, and the **Agents** are designed to perform the work
- When build is launched, Jenkins controller connects to the specified agent, and runs all actions on it while fetching output back to the controller (master).
- If there are any system dependencies for your build to work (Git, Maven, etc.), they need to be satisfied on the agents, even though the build should be designed to be self-contained.
- Can be connected via several available methods:
  - SSH (SSH based, master to slave connection)
  - TCP/WebSocket (Agent to Controller connection)

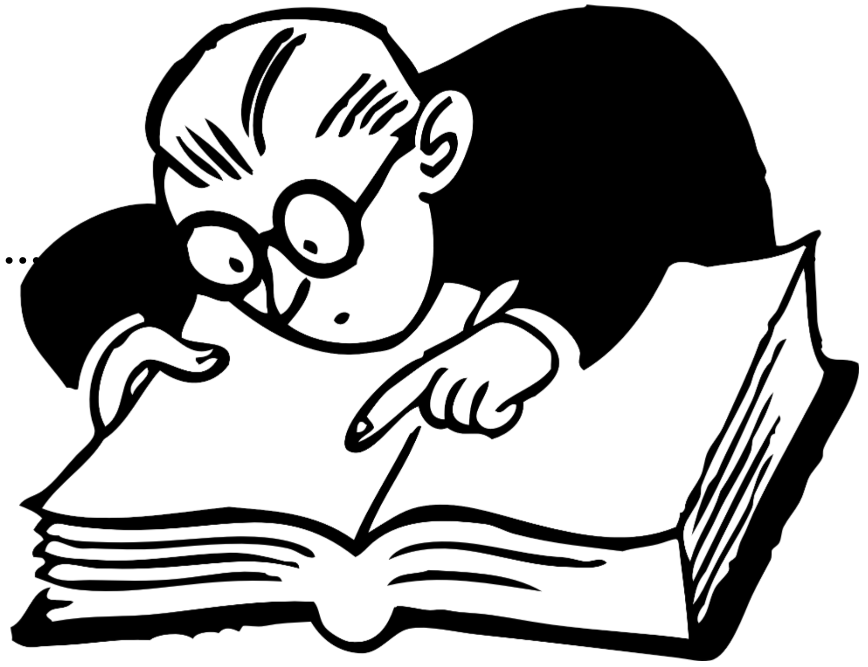


- Agents can be assigned to a label. Assigning labels to jobs instead of agent names is great for achieving redundancy (if one agent is down, other under the label will be used).

TERMINOLOGY

- **Agent = Node**
  - agents are computers that are set up to build projects for a controller. Jenkins runs a separate program called "slave agent" on nodes. When agents are registered to a controller, the controller starts distributing loads to this agent.
- **Project = Job**
  - A runnable task that is controlled/monitored by Jenkins
- **Upstream job**
  - A job can have one or several upstream job, which means that a build for the current job may be scheduled when an upstream build is finished. Every upstream build can schedule a build in the downstream job, but there are several options and plugins which can customize this behavior.
- **Downstream job**
  - A job can have one or several downstream jobs. The current project is then known as an upstream job of the downstream job.
- **Publisher**
  - A publisher is part of the post-build actions. A publisher may report stable or unstable result depending on the result of its processing.

TERMINOLOGY



- Jenkins **instance** = Jenkins installation

- **Artifact**
  - An immutable file generated during a Build or Pipeline run which is archived onto the Jenkins controller (master) for later retrieval by users
- **Workspace:**
  - Disposable directory on Node used as a working directory for building
- **Executor:**
  - Separated stream of builds to be run on Node in parallel. Node can have 1 or more Executors. Special executors can be created dynamically (one-off executors) to run lightweight jobs used mostly for orchestration purposes.

TERMINOLOGY

- **Stable build**
  - A build is stable if it was built successfully and no publisher reports it as unstable.
- **Unstable build**
  - A build is unstable if it was built successfully and one or more publishers report it unstable.
- **Broken/Failed build**
  - A build is broken if it failed during building. That is, it is not successful.

INSTALLATION

- Use provided installation guide in the repo (Setting up your lab environment)
- Example for Ubuntu:
    - **Prerequisites - Install JAVA:**
        - Jenkins requires Java in order to run (OpenJDK or Oracle JDK). **Java 8/11 is the ONLY** supported

- **Start/Stop Jenkins**
    - vagrant@jenkins:~$ sudo /etc/init.d/jenkins
    - Usage: /etc/init.d/jenkins {start|stop|status|restart|force-reload}

INSTALLATION

- **Access Jenkins** at : http://ip_address_or_domain_name:8080
- Initial Jenkins Web UI **username** is **admin** with **password** stored at /var/lib/jenkins/secrets/initialAdminPassword
- **Jenkins home** is kept at /var/lib/jenkins/. Can be overridden by setting JENKINS_HOME variable before launching Jenkins
- **Default port:** 8080. Can be overridden by setting JENKINS_PORT variable before launching Jenkins.
- **Jenkins WAR** is kept at /usr/share/jenkins/jenkins.war
- **Jenkins log file:** /var/log/jenkins/jenkins.log
- **System configuration file**: /etc/default/jenkins
- **Jenkins configuration file**: /var/lib/jenkins/config.xml
- **Jenkins system user** is jenkins:jenkins. Never run Jenkins server as root!

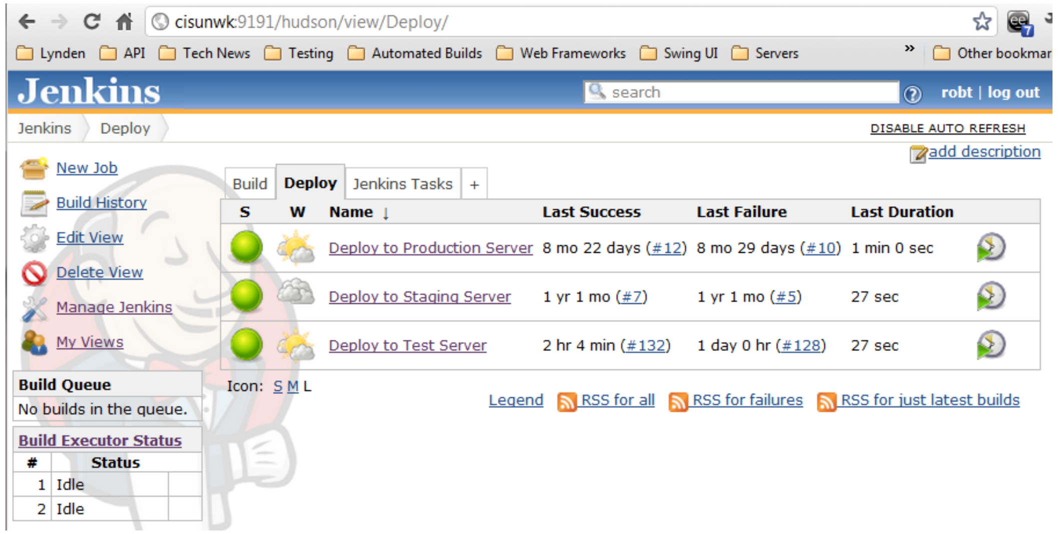
JOBS CONFIGURATION

- **Actions performed by a job:**
  - pull code from version control system (ex GIT)
  - copies files from other job
  - run commands/scripts (Bash/Batch/Groovy/Ruby etc.), builds application
  - run tests
  - job execution can be scheduled periodically (hourly, daily, weekly..) or can be triggered by an external hook (when a GIT commit is pushed)
  - post-build steps include generating reports, trigger another job, send notifications and results (ex email)

  Make sure to always have builds discard policy enabled for each job to prevent filling up the disk

- **Most popular type of jobs are:**
  - Free-style software project
  - Pipeline

JOBS CONFIGURATION


cisunwk:9191/hudson/view/Deploy/

Lynden | API | Tech News | Testing | Automated Builds | Web Frameworks | Swing UI | Servers | Other bookmar

Jenkins | search | robt | log out

Jenkins > Deploy | DISABLE AUTO REFRESH

add description

New Job
Build History
Edit View
Delete View
Manage Jenkins
My Views

Build | Deploy | Jenkins Tasks | +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
●		Deploy to Production Server	8 mo 22 days (#12)	8 mo 29 days (#10)	1 min 0 sec	
●		Deploy to Staging Server	1 yr 1 mo (#7)	1 yr 1 mo (#5)	27 sec	
●		Deploy to Test Server	2 hr 4 min (#132)	1 day 0 hr (#128)	27 sec	

Icon: S M L

Legend | RSS for all | RSS for failures | RSS for just latest builds

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

- **How can we create Jenkins jobs?**

1. Using the Jenkins Interface ->New Item
2. Writing Domain Specific Language (DSL) scripts:
    - https://jenkinsci.github.io/job-dsl-plugin/#
    - allows users to describe jobs using a Groovy-based language
    - can be pushed to GIT
3. Using Ansible & xml/jinja : https://docs.ansible.com/ansible/2.6/modules/jenkins_job_module.html

JOBS CONFIGURATION

Freestyle vs Pipeline Projects

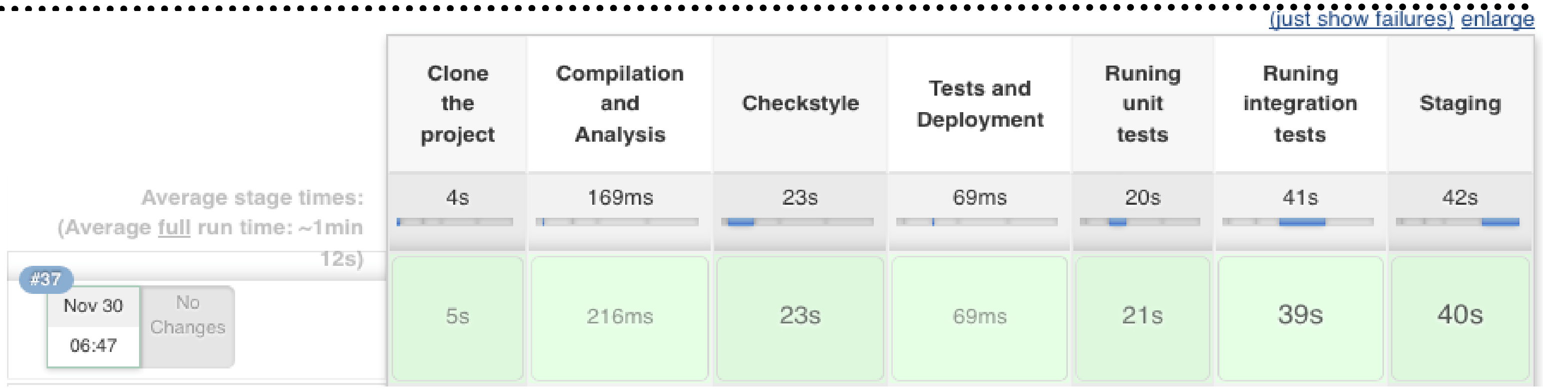
Freestyle:

- Old way of Jenkins job configuration using sequential tasks for defining an application lifecycle.
- Convenient for defining simple jobs (i.e. executing a shell script).
- Additional steps can be introduced through various plugins.
- Configured through UI, so they are more difficult to maintain when compared to Pipeline plain-text definitions.

Pipeline:

- New way of defining Jenkins jobs by specifying the whole application lifecycle in a plain-text file.
- Support for complex real-world CD requirements (e.g. parallel execution, looping, joining).
- Extensible using plugins and Pipeline DSL based on Groovy.
- Can survive restarts of Jenkins controller.
- Can be resumed from any of several saved checkpoints.

JOBS CONFIGURATION



(just show failures) enlarge

	Clone the project	Compilation and Analysis	Checkstyle	Tests and Deployment	Runing unit tests	Runing integration tests	Staging
Average stage times: (Average full run time: ~1min 12s)	4s	169ms	23s	69ms	20s	41s	42s
#37 Nov 30 06:47 No Changes	5s	216ms	23s	69ms	21s	39s	40s

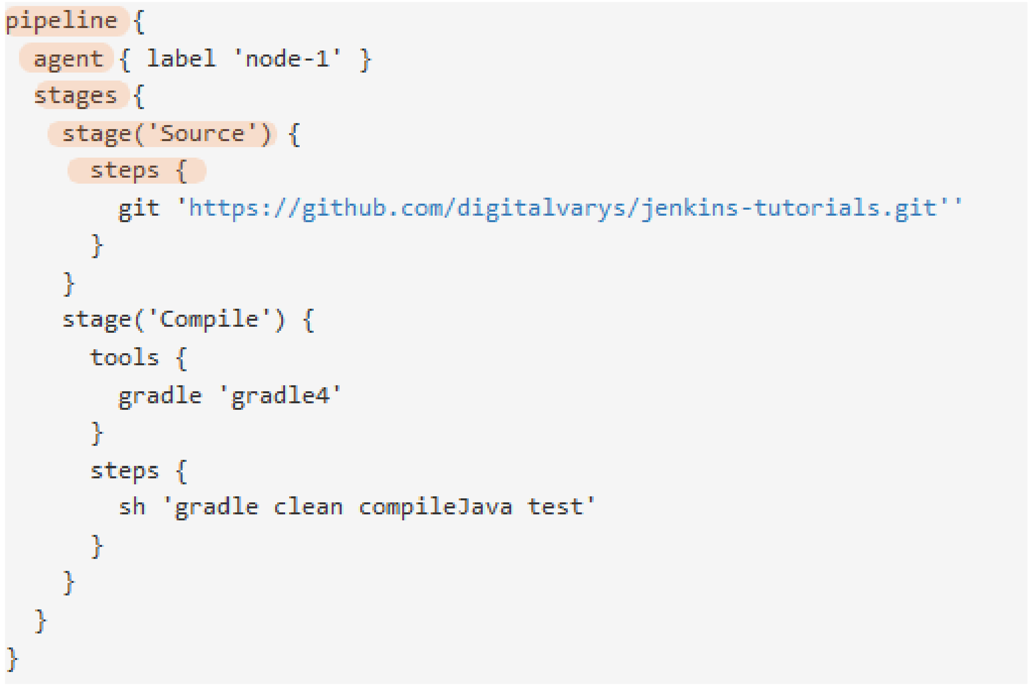
Pipeline:

- A pipeline has multiple stages that can be executed sequential or in parallel
- Advantages:
  - **Code :** Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline ("Pipeline-as-code“)
  - **Durable:** Pipelines can survive both planned and unplanned restarts of the Jenkins master
  - **Pausable:** Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run
  - **Versatile**: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
  - **Extensible**: The Pipeline plugin supports custom extensions to its DSL and multiple options for integration with other plugins

DECLARATIVE VS SCRIPTED PIPELINE

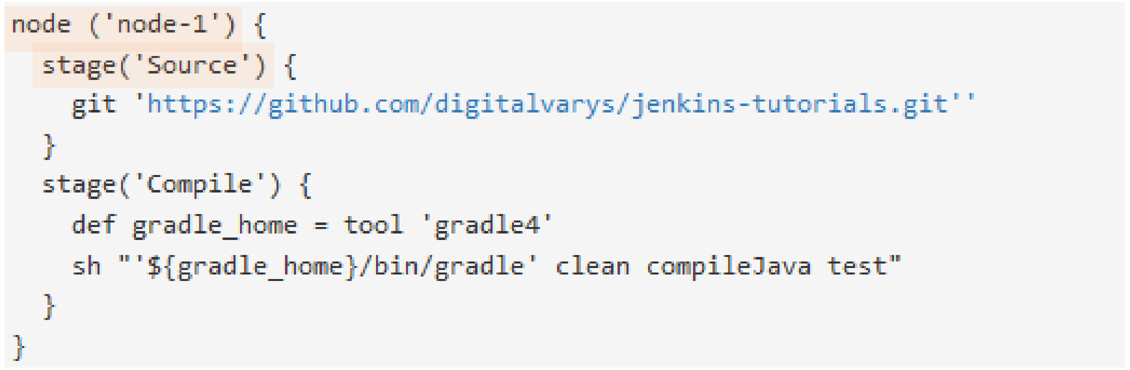
The **key difference** between Declarative pipeline and Scripted pipeline would be with respect to their **syntaxes** and their **flexibility**.

DECLARATIVE



```
pipeline {
  agent { label 'node-1' }
  stages {
    stage('Source') {
      steps {
        git 'https://github.com/digitalvarys/jenkins-tutorials.git''
      }
    }
    stage('Compile') {
      tools {
        gradle 'gradle4'
      }
      steps {
        sh 'gradle clean compileJava test'
      }
    }
  }
}
```

SCRIPTED



```
node ('node-1') {
  stage('Source') {
    git 'https://github.com/digitalvarys/jenkins-tutorials.git''
  }
  stage('Compile') {
    def gradle_home = tool 'gradle4'
    sh "'${gradle_home}/bin/gradle' clean compileJava test"
  }
}
```

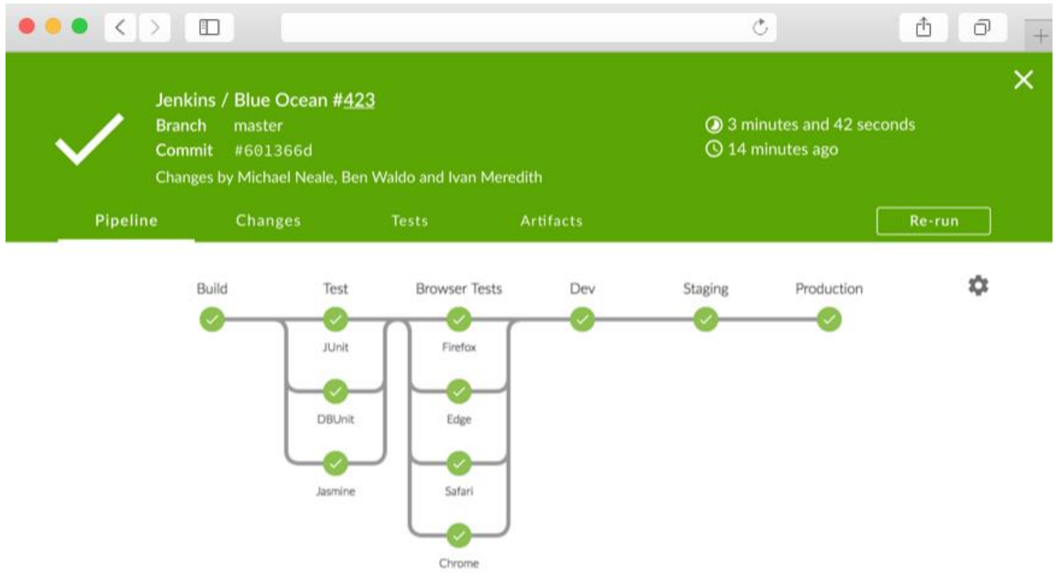
JENKINS

"Blue Ocean" Interface

- Jenkins 2.0 introduced a very intuitive and modern web interface called "Jenkins Blue Ocean".

  https://www.jenkins.io/projects/blueocean/

- It has a good visual representation of all the pipelines and allows developers to define and visually represent pipelines in a way that is easy to understand.
- Requires installing the "Blue Ocean" plugin.



Jenkins / Blue Ocean #423
Branch master
Commit #601366d
Changes by Michael Neale, Ben Waldo and Ivan Meredith
3 minutes and 42 seconds
14 minutes ago

THE JENKINSFILE

- You can define a Pipeline in either the web UI or with a “Jenkinsfile”.
- Complex Pipelines are difficult to write and maintain within the classic UI’s “Script” text area of the Pipeline configuration page.
- You can use your favorite IDE or text editor to write the Pipeline and save it under the root directory of your project, next to the application code that Jenkins is supposed to build.
- The Jenkinsfile is a plain-text file that contains the definition of a Jenkins Pipeline.
- Creating a Jenkinsfile and checking it into the source control repository is generally considered a best practice.
- Supported by popular IDEs: https://www.jenkins.io/doc/book/pipeline/development/

WEBHOOKS

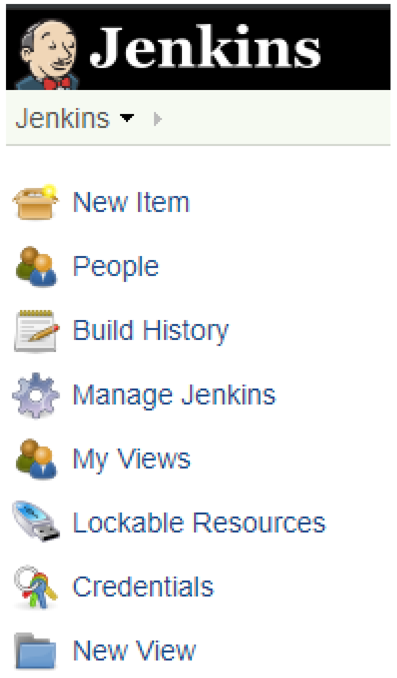
- A webhook is a mechanism to automatically trigger the build process of a Jenkins project whenever there is a new change.
- Through Webhooks, Git repo can send a “PUSH” notifications to Jenkins whenever a new commit gets pushed to a remote Git repository, so there are no disadvantages of polling.
- In order for builds to be triggered automatically, besides configuring Jenkins, you must also configure a Webhook under the settings of your Git repository.
- Read the Webhook documentation for your repository hosting service of choice

CREDENTIALS

Credentials configured in Jenkins are stored in an encrypted form on the master Jenkins instance

Jenkins can store the following types of credentials:

- Secret text - a token such as an API token (e.g. a GitHub personal access token),
- Username and password - which could be handled as separate components or as a colon separated string in the format username:password
- Secret file - which is essentially secret content in a file
- SSH Username with private key - an SSH public/private key pair
- Certificate - a PKCS#12 certificate file and optional password
- Docker Host Certificate Authentication credentials



PLUGINS

- Plugins represent the Jenkins superpowers compared to competition as they bring a number of features and integrations for different purposes
- Anyone can also create and use their own plugins that match specific needs of their project
- Some useful plugins:
  - Pipeline
  - Job Configuration History
  - Parameterized Trigger
  - Job DSL
  - Environment Injector
  - Timestamper
  - Workspace Cleanup
  - Docker Slaves
  - Amazon EC2

RESOURCES

https://jenkins.io/doc/

https://www.safaribooksonline.com/library/view/mastering-jenkins/9781784390891/

https://app.pluralsight.com/library/courses/jenkins-2-getting-started/table-of-contents

https://www.tutorialspoint.com/jenkins/index.htm

USEFUL LINKS

[Teams discussion channel][Feedback form][GitLab]

THANK YOU

MARILENA ISTRATE

Marilena.Istrate@endava.com