ENTERPRISE SYSTEMS INTEGRATION

ACIT4850 – WINTER 2024

AGENDA – LESSON 6

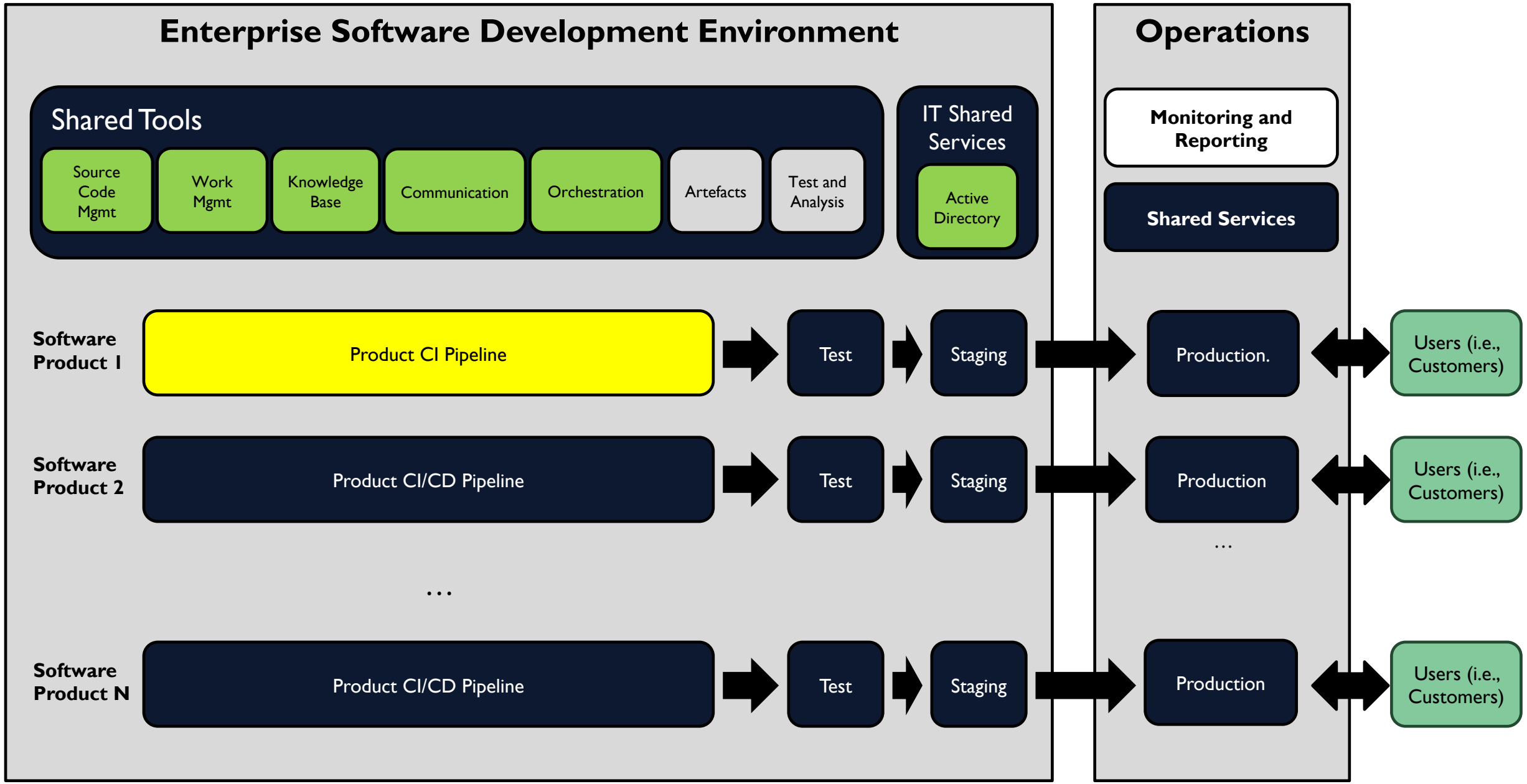
- Quick Review
- Quiz 5 on D2L
- Topics
  - Jenkins Pipeline – Replay
  - Jenkins Pipeline – Shared Libraries
- Lab Requirements
- Lab
  - Demo your Lab 5 (if you haven't already)
  - Start on Lab 6 – Due by end of next class

QUICK REVIEW

- What language do the Shared Libraries support in Jenkins pipelines?
- Why would we want to use Shared Libraries in Jenkins pipelines? What was the acronym used for this?
- What tools/resources are available to help us with the development of Jenkins pipelines?
- How do we make Shared Libraries available to pipeline jobs in Jenkins?

QUIZ 5

- On Jenkins pipeline Shared Libraries and Testing
- On the Learning Hub (aka D2L), Open Book
- Your have 15 minutes to complete it


Enterprise Software Development Environment
Shared Tools
Source Code Mgmt
Work Mgmt
Knowledge Base
Communication
Orchestration
Artefacts
Test and Analysis
IT Shared Services
Active Directory
Operations
Monitoring and Reporting
Shared Services
Software Product 1
Product CI Pipeline
Test
Staging
Production.
Users (i.e., Customers)
Software Product 2
Product CI/CD Pipeline
Test
Staging
Production
Users (i.e., Customers)
...
...
Software Product N
Product CI/CD Pipeline
Test
Staging
Production
Users (i.e., Customers)

THE ROADMAP (AKA COURSE SCHEDULE)

Week	Topics	Notes
1	• Components of an Enterprise Development Environment • Software Source Code Management	Lab 1
2	• Work Management and Knowledge Base Tools	Lab 2, Quiz 1
3	• Tool Selection – Requirements • Integration and Security	Lab 3, Quiz 2
4	• Tool Selection – Stakeholders/Process • Continuous Integration (CI) Tool • CI Tool Setup	Lab 4, Quiz 3
5	• CI Pipelines – Python	Lab 5, Quiz 4
6	• CI Pipelines – Shared Libraries	Lab 6, Quiz 5, Assignment 1 Due
7	• CI Pipelines – Java and Static Code Analysis *Note: At home lab for Monday set*	Lab 7, Quiz 6 (Sets A and B)
8	• Midterm	Midterm Review Quiz
9	• CI Pipelines – Alternate Tools	Lab 8, Quiz 6 (Set C), Quiz 7
10	• Spring Break	
11	• CI Pipelines – Artifact Management (Java)	Lab 9, Quiz 8, Assignment 2 Due
12	• Continuous Delivery (CD) • CD Pipelines - Containerization	Lab 10, Quiz 9
13	• CD Pipelines – Deployment • Developer Workflows *Note: At home lab for Monday Set*	Lab 11, Quiz 10 (Sets A and B)
14	• Microservices Pipelines • Final Exam Preview	Quiz 10 (Set C), Assignment 3 Due
15	**Final Exam**	

TESTING JENKINS PIPELINES

- **Most Basic** - Check in your changes to the pipeline and re-run in Jenkins. This can create a lot of extra checkins as your are troubleshooting.

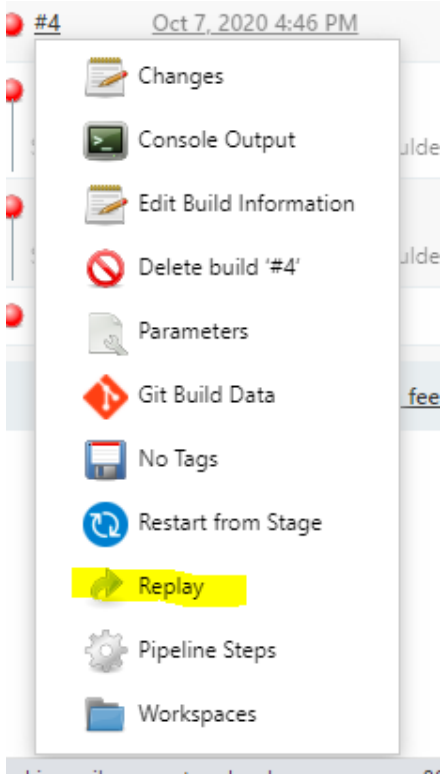
- **Replay in a Jenkins Job** – Adjust the Jenkinsfile and re-run the job. Good for initial pipeline development.
  - Limitation – doesn't work well for shared libaries

- **Blue Ocean Editor** – In Jenkins. More of a more modern UI in Jenkins that's focused on pipelines.

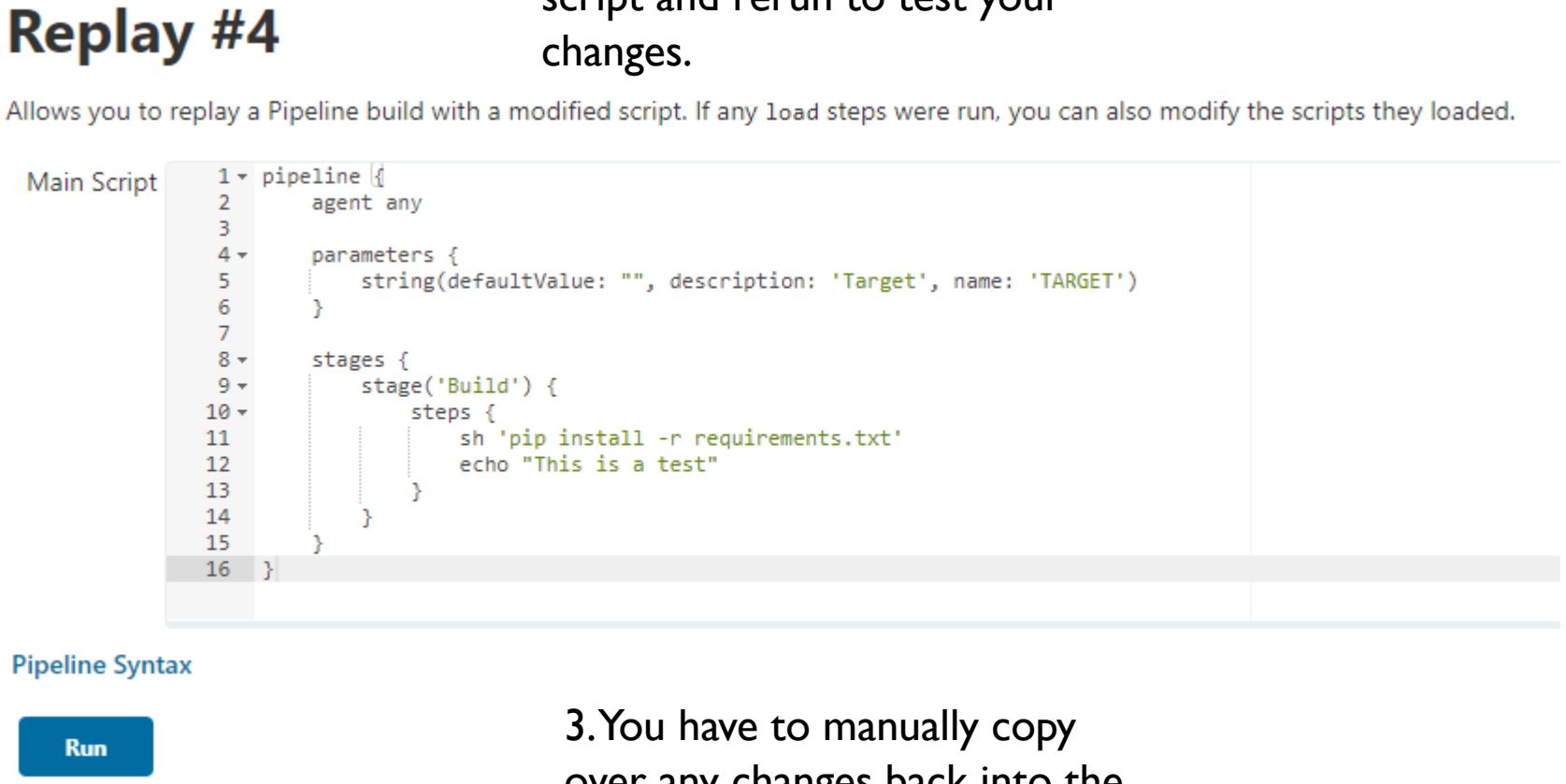
- **Lint** – A basic syntax checker

EXAMPLE - REPLAY

1. Select Replay for one of the previous builds



2. You can edit the pipeline script and rerun to test your changes.



3. You have to manually copy over any changes back into the Jenkinsfile in your Git repo

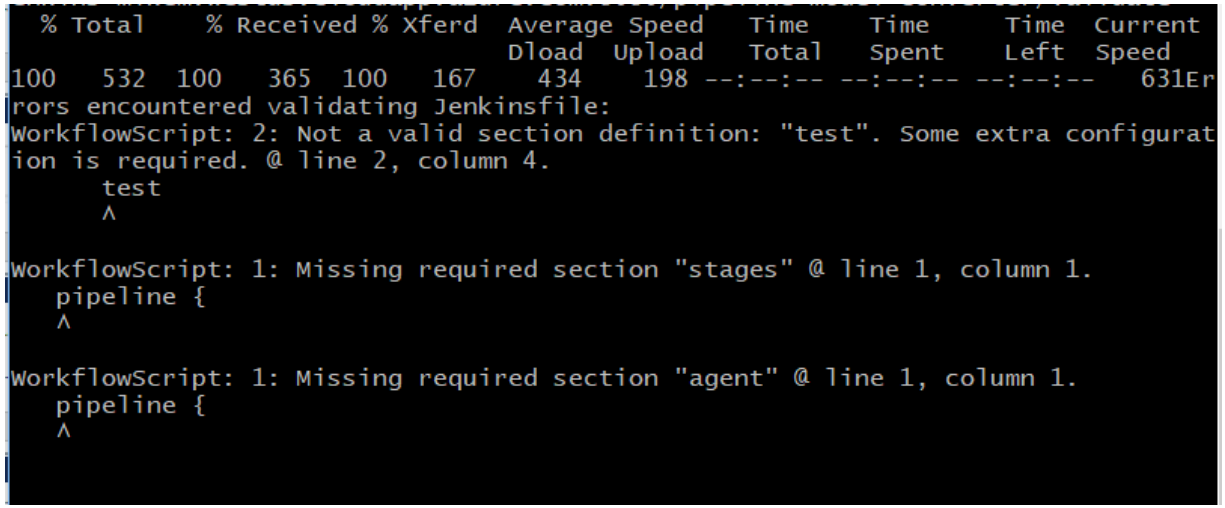
EXAMPLE - LINT

- There is an endpoint in your Jenkins server that you can use to validate your Jenkinsfile

curl --user username:password -X POST -F "jenkinsfile=<Jenkinsfile" JenkinsURL/pipeline-model-converter/validate

Where:

- username – valid user in your Jenkins server
- password – corresponding user password
- Jenkinsfile – the name of you local Jenkinsfile (should be in the same folder as you run the curl command)
- JenkinsURL – the URL of your Jenkins server



SHARED LIBRARIES

You can create a Shared Library for a Groovy function that may be used in a Jenkins pipeline.

You can also create a Shared Library that defines an entire pipeline. Often times many repositories have the same type and structure of code, so the same pipeline applies. This is especially true for microservices type projects where there are many small applications that need to be built, often in the same language.

Shared Library Repository

libs – Groovy source files

vars – This is where the shared pipelines. They are exposed as variables in Jenkins pipelines.

resources – Other files

At minimum, your Shared Library Repository will have a vars folder.

NEW LAB REQUIREMENTS

- (Existing) **REQ1150** – The Enterprise Development Environment shall automatically trigger a Continuous Integration Pipeline on a Software Project (i.e., repository) in Source Code Management when the source code changes. Note: A push to a project in GitLab should trigger the corresponding build job.
- (Existing) **REQ1160** – The Enterprise Development Environment shall support Continuous Integration Pipelines for Python projects. At minimum, the pipeline will include build, test, packaging and artifact storage.
- (New) **REQ1170** – The Enterprise Development Environment shall support re-use of CI Pipeline definitions across similar projects. For example, similar Python projects (i.e., Flask applications) should be able to use the same pipeline definition.

NEW LAB REQUIREMENTS

- **REQ1150** – The Enterprise Development Environment shall automatically trigger a Continuous Integration Pipeline on a Software Project (i.e., repository) in Source Code Management when the source code changes. Note: A push to a project in GitLab should trigger the corresponding build job.

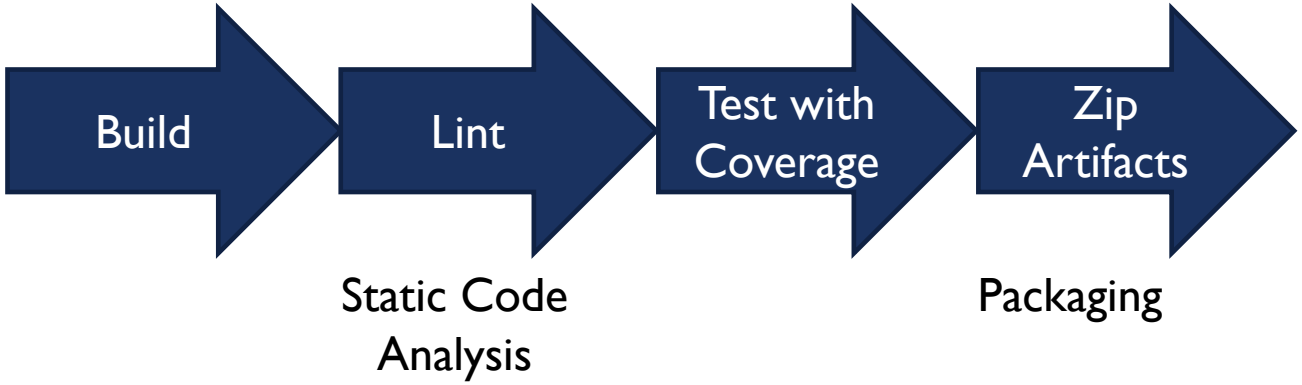
You will need to do this once again for a new project.

NEW LAB REQUIREMENTS

- **REQ1160** – The Enterprise Development Environment shall support Continuous Integration Pipelines for Python projects. At minimum, the pipeline will include build, test, packaging and artifact storage.

You will be modifying your Jenkins pipeline to add a Lint and Packaging stages.

You will be modifying your existing Test stage to work on any test files, display coverage and remove the API Test stage.



NEW LAB REQUIREMENTS

- **REQ1170** – The Enterprise Development Environment shall support re-use of CI Pipeline definitions across similar projects. For example, similar Python projects (i.e., Flask applications) should be able to use the same pipeline definition.

After modifying your Jenkins pipeline for your existing Python application, you will make a Shared Library from that pipeline. You will re-use it in another similar Python application.

You will put your Shared Library in a separate Git repo called ci_functions.

VIRTUAL ENVIRONMENT

- Use a Virtual Environment for the Build and Test stages of the Pipeline
  - Install the Virtual Environment module in your Docker image (so it is available to all pipelines we create)
  - Create and Activate the Virtual Environment in your Build stage (or create a new Setup stage if you want)
    - The Build stage should install the requirements.txt in the Virtual Environment
    - The Test should should run the tests in the Virtual Environment
    - You'll likely need to install the coverage dependency in the Virtual Environment for the tests (not ideal, but okay)
- The Virtual Environment will prevent the Jenkins container from being "polluted" with dependencies from all the pipelines being run on that Jenkins Master instance.

PYTHON LINT

- You will be using pylint-fail-under for the Lint stage of your shared pipeline
- The syntax is:

```
pylint --fail-under <min passing score> <file or wildcard>
```

  - min passing score is a number, like 5.0
  - file or wildcard can be something like *.py to check all Python files

PIPELINE UTILITY STEPS

- For part of the lab, you fill need to find all files that start with test. You can use the findFiles from Pipeline Utility Steps:
  - You need to install the Plugin in Jenkins (Manage Jenkins -> Plugins)
  - Then you can use fileFiles in your pipeline:

```
def files = findFiles(glob: "<Your file pattern here>")
```

Ref: https://Jenkins.io/doc/pipeline/steps/pipeline-utility-steps

  - The files variable will be a list of matching filenames. You can loop through those files:

```
for (file in files) {
    // Do something – file.path gives you the filepath
}
OR
for (int i = 0; i < files.size(); i++) {
     // Do something – files[i].path gives you the file path
}
```

Groovy code in your pipeline should be in a script { } block

If you reference a variable in a sh command, you must use double quotes.

```
sh '${file}' -> won't work
sh "${file}" -> works
```

COVERAGE

- For part of the lab, you want to generate coverage reports for the code. Coverage is the amount of code covered by the tests.
- You want to run coverage on each unit test in your repo:
  - `coverage run --omit */dist-packages/*,*/site-packages/* <test file path>`

  This will determine the coverage for the test, excluding any Python dependencies used
- After you've run coverage on each unit test, generate the coverage report:
  - `coverage report`

  This will generate the coverage report on the console

CLEANING UP UNIT TEST RESULTS

In the Test and Coverage stage, we will run this code before running the tests:

```
// Remove any existing test results
script {
    def test_reports_exist = fileExists 'test-reports'
    if (test_reports_exist) {
        sh 'rm test-reports/*.xml || true'
    }

    def api_test_reports_exist = fileExists 'api-test-reports'
    if (api_test_reports_exist) {
        sh 'rm api-test-reports/*.xml || true'
    }
}
```

And this code after running the tests:

```
// Process the test results if they exist
script {
    def test_reports_exist = fileExists 'test-reports'
    if (test_reports_exist) {
        junit 'test-reports/*.xml'
    }

    def api_test_reports_exist = fileExists 'api-test-reports'
    if (api_test_reports_exist) {
        junit 'api-test-reports/*.xml'
    }
}
```

PACKAGE – ZIP ARTIFACTS

- There is a plugin in Jenkins that allows you to zip files, but some had problems with it.
- An alternative is to use zip directly.
- In the Dockerfile for your Jenkins installation, add zip:
  - apt-get -y install zip
- Then in the Package stage of your pipeline, you can zip up all the .py files in a shell command:
  - zip app.zip *.py
  - You will still need to make app.zip an "artifact" in Jenkins

CALLABLE FUNCTION IN SHARED LIBRARY

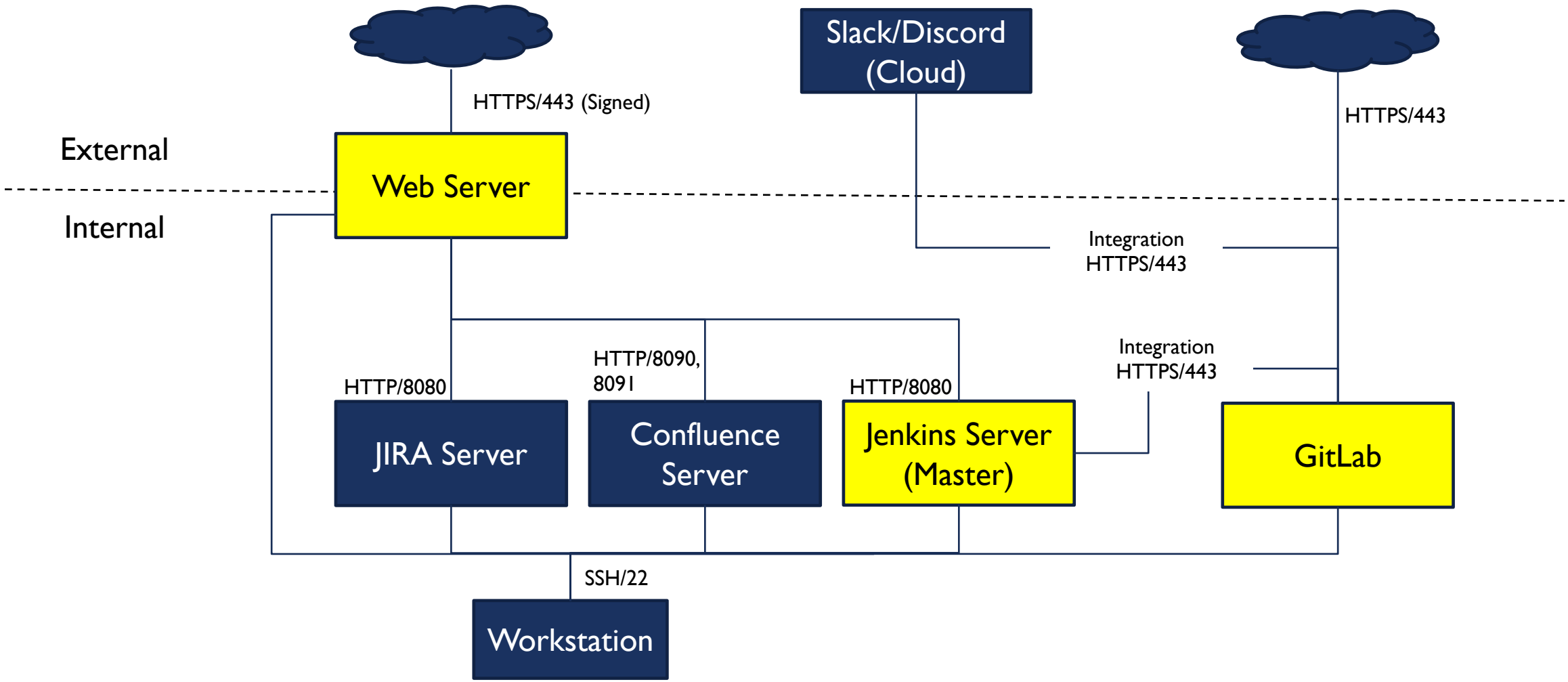
- Wrap your re-usable pipeline in the following in your shared library file (i.e., python_build.groovy):

```
def call() {
    // Your pipeline definition here
}
```

- Then in your Jenkinsfile you can use it as follows:

```
@Library('ci_functions@main') _
python_build()
```

YOUR ENTERPRISE DEVELOPMENT ENVIRONMENT (SO FAR)



TODAY'S LAB

1. Demo Lab 5 Before the End of Class
2. Start on Lab 6
    1. You will do this together with your partner
    2. Demo by end of class next week
3. Next week we will work on Java builds and a source code analysis tool