ENTERPRISE SYSTEMS INTEGRATION

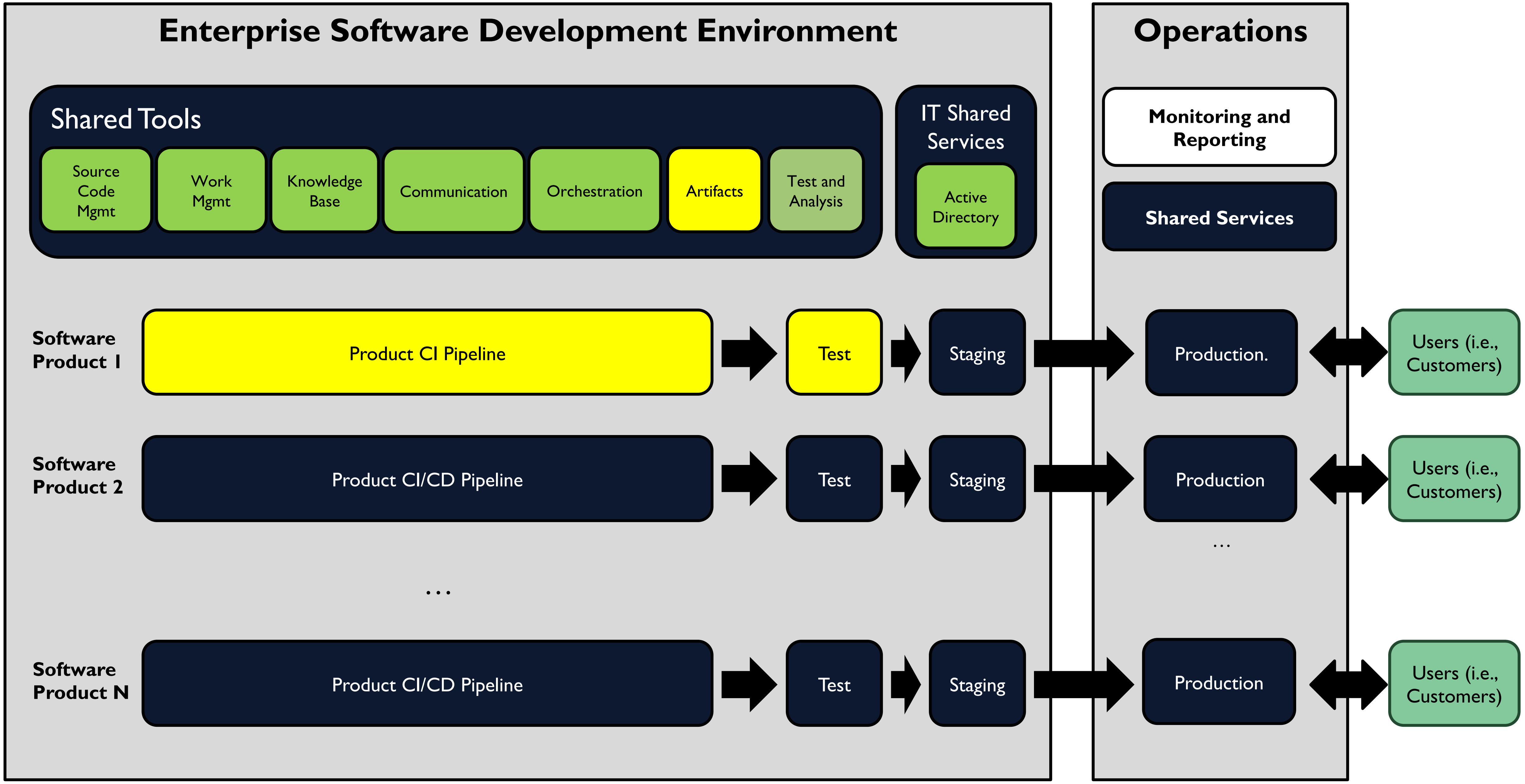
ACIT4850 – WINTER 2024

AGENDA – LESSON 10

- Quiz 9 on D2L
- What's Left?
  - Labs, Assignments and Final
- Topics
  - CI/CD Pipelines
  - Containerization – Docker
- Lab 9 – Containerization

QUIZ 9

- Open Book – You should have the Lesson 10 reading materials available
- You have 15 minutes to complete
- We will resume in 15 minutes, or when everyone is done


Enterprise Software Development Environment
Shared Tools
Source Code Mgmt
Work Mgmt
Knowledge Base
Communication
Orchestration
Artifacts
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 (GitLab CI)	Lab 8, Quiz 6 (Set C), Quiz 7
10	• Spring Break	
11	• Continuous Delivery (CD) • CD Pipelines - Containerization	Lab 9, Quiz 8, Assignment 2 Due
12	• CD Pipelines – Deployment • Developer Workflows	Lab 10, Quiz 9
13	• CD Pipelines – Alternate Tools (GitLab CI) • *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**	

LABS – LAST LABS LEFT

Lab 10 – This Week

- Add a Containerization Stage to your build pipelines
- Due end of next class (demo)

Lab 11 - Next Class

- Simple Deployment of Docker Images
- Exercise a Merge Request

Lab 12

- Repeat the above on GitLab CI

ASSIGNMENTS AND FINAL EXAM

Assignment 3

- Combines 3855 and 4850 (i.e., pipelines for your services)

Final Exam

- In Person – Week of April 15th
- "Written" and "Practical" components

ASSIGNMENT 1

- Key Points:
  - Make sure you know your Stakeholders and identify their requirements
  - Understand your audience and make sure your assessment is easy to review:
    - Requirements assessed are clearly identified
    - Comparison makes it easy to determine which products meet the requirements

In the Final Exam, you may have a question asking you to identify the requirements for a tool assessment from the perspective of a particular stakeholder based on a brief scenario.

ASSIGNMENT 1 – EXAMPLE REQUIREMENTS

Mandatory Features

- **REQ1010:** The source code management tool shall support Active Directory or LDAP
- **REQ1020:** The source code management tool shall allow for either on-premise or cloud infrastructure production deployment
- **REQ1030:** The source code management tool shall be Git based.
- **REQ1040:** The source code management tool shall be able to fully transfer the existing subversion repository history to the new SCM tool.

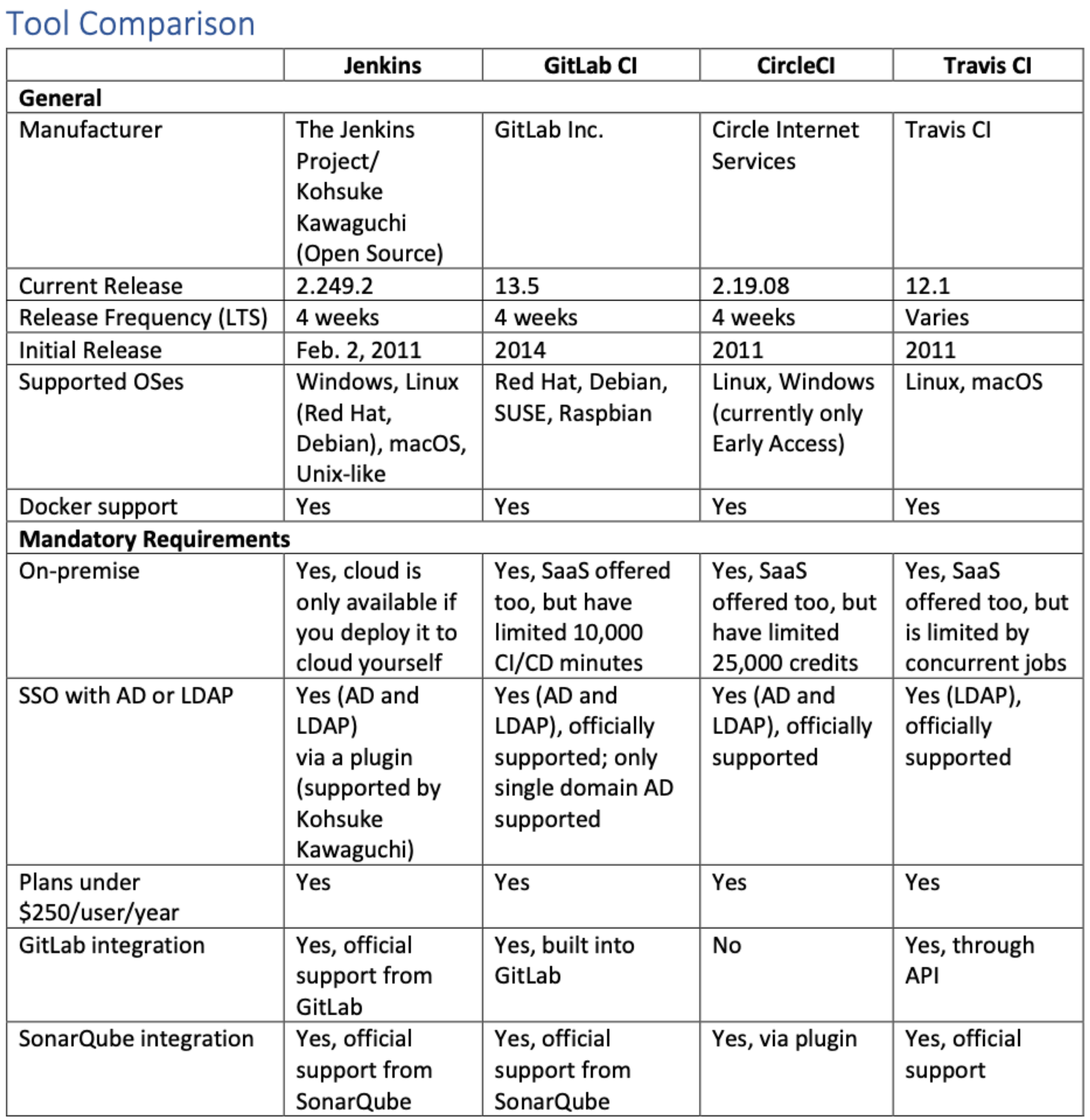
Optional Features

- **REQ2010:** The source code management tool shall allow logical grouping of source code repositories into products
- **REQ2020:** The source code management tool shall support gigabytes of large binary files.
- **REQ2030:** The source code management tool shall have an average annual cost of $250 USD or less per developer
- **REQ2040:** The source code management tool shall provide dedicated in-house support.
- **SEC1010:** The source code management tool shall have a static source code analysis capability for code quality and security; either from the SCM tool or from 3rd party integration.
- **SEC1020:** The source code management tool shall allow user permission repository restrictions based on their current working product.

Describes the requirements simply for quick review

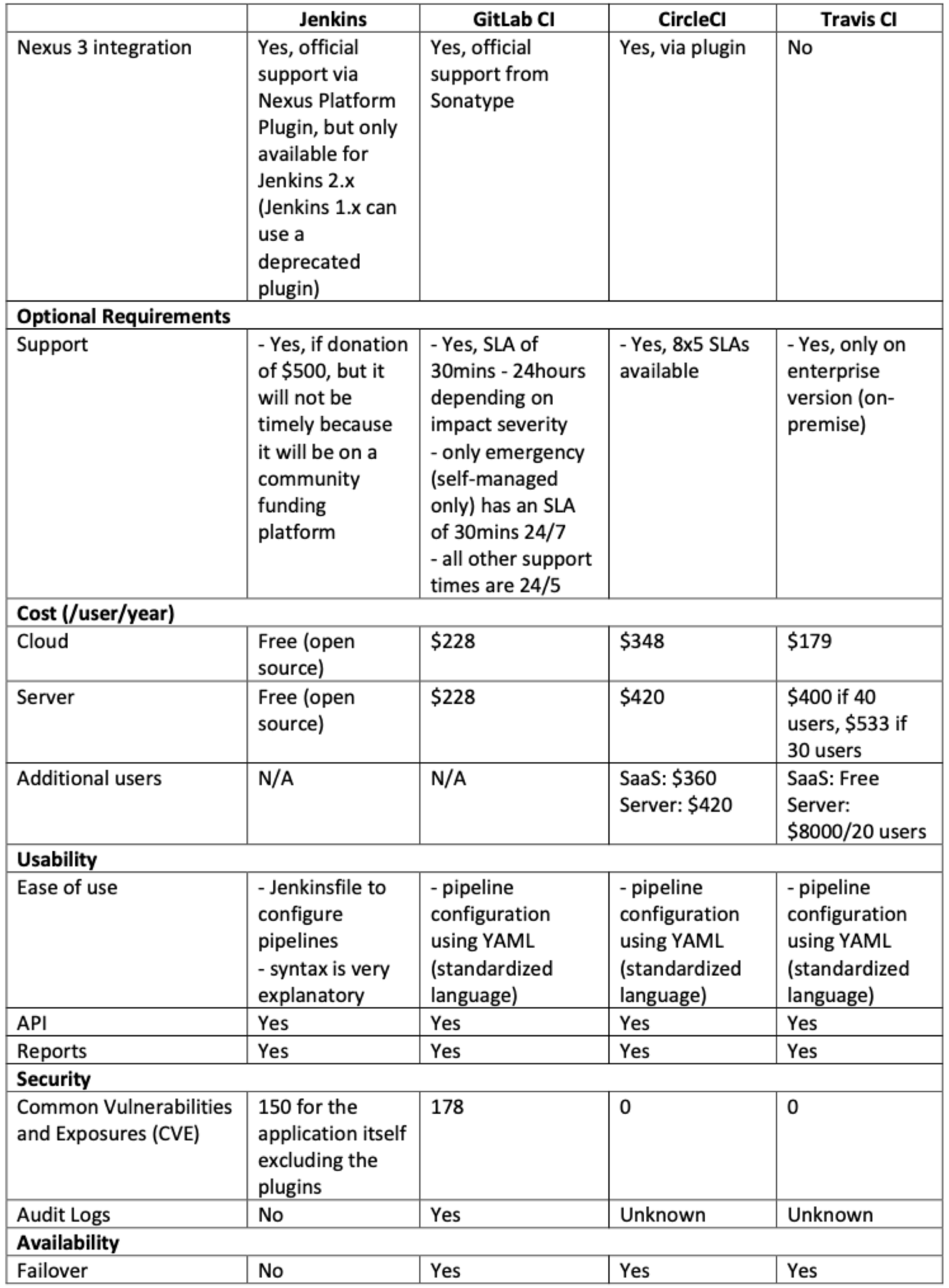
Organized by Mandatory and Optional

Numbered for Traceability



Tool Comparison

	Jenkins	GitLab CI	CircleCI	Travis CI
General				
Manufacturer	The Jenkins Project/ Kohsuke Kawaguchi (Open Source)	GitLab Inc.	Circle Internet Services	Travis CI
Current Release	2.249.2	13.5	2.19.08	12.1
Release Frequency (LTS)	4 weeks	4 weeks	4 weeks	Varies
Initial Release	Feb. 2, 2011	2014	2011	2011
Supported OSes	Windows, Linux (Red Hat, Debian), macOS, Unix-like	Red Hat, Debian, SUSE, Raspbian	Linux, Windows (currently only Early Access)	Linux, macOS
Docker support	Yes	Yes	Yes	Yes
Mandatory Requirements				
On-premise	Yes, cloud is only available if you deploy it to cloud yourself	Yes, SaaS offered too, but have limited 10,000 CI/CD minutes	Yes, SaaS offered too, but have limited 25,000 credits	Yes, SaaS offered too, but is limited by concurrent jobs
SSO with AD or LDAP	Yes (AD and LDAP) via a plugin (supported by Kohsuke Kawaguchi)	Yes (AD and LDAP), officially supported; only single domain AD supported	Yes (AD and LDAP), officially supported	Yes (LDAP), officially supported
Plans under $250/user/year	Yes	Yes	Yes	Yes
GitLab integration	Yes, official support from GitLab	Yes, built into GitLab	No	Yes, through API
SonarQube integration	Yes, official support from SonarQube	Yes, official support from SonarQube	Yes, via plugin	Yes, official support



	Jenkins	GitLab CI	CircleCI	Travis CI
Nexus 3 integration	Yes, official support via Nexus Platform Plugin, but only available for Jenkins 2.x (Jenkins 1.x can use a deprecated plugin)	Yes, official support from Sonatype	Yes, via plugin	No
Optional Requirements				
Support	- Yes, if donation of $500, but it will not be timely because it will be on a community funding platform	- Yes, SLA of 30mins - 24hours depending on impact severity - only emergency (self-managed only) has an SLA of 30mins 24/7 - all other support times are 24/5	- Yes, 8x5 SLAs available	- Yes, only on enterprise version (on-premise)
Cost (/user/year)				
Cloud	Free (open source)	$228	$348	$179
Server	Free (open source)	$228	$420	$400 if 40 users, $533 if 30 users
Additional users	N/A	N/A	SaaS: $360 Server: $420	SaaS: Free Server: $8000/20 users
Usability				
Ease of use	- Jenkinsfile to configure pipelines - syntax is very explanatory	- pipeline configuration using YAML (standardized language)	- pipeline configuration using YAML (standardized language)	- pipeline configuration using YAML (standardized language)
API	Yes	Yes	Yes	Yes
Reports	Yes	Yes	Yes	Yes
Security				
Common Vulnerabilities and Exposures (CVE)	150 for the application itself excluding the plugins	178	0	0
Audit Logs	No	Yes	Unknown	Unknown
Availability				
Failover	No	Yes	Yes	Yes

What's good about this comparison?

How could we make it better?

ASSIGNMENT 2

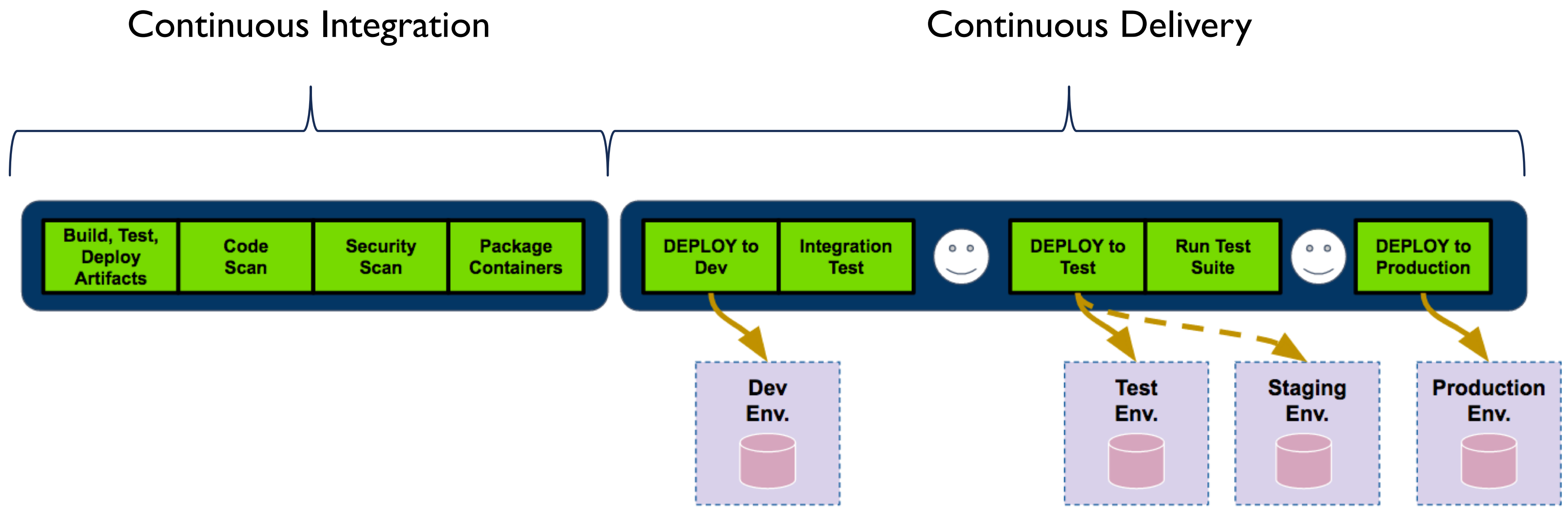
- Due March 24th
- How would you compare Jenkins and GitLab CI?

CI/CD DEFINITIONS

- Continuous Integration – Includes the following:
  - Build
  - Unit level tests (those that do not require a deployment)
  - Static code analysis
  - Packaging (i.e., zip or container)
  - Push to an Artifact Repository
- Continuous Delivery
  - Getting code into a deployable state (you can argue that Packaging/Artifact Management is part of CD rather than CI)
  - Automation of the deployment of that deployable code into internal environments
    - Development, Test, Staging, Production

CI/CD PIPELINE

To enable Continuous Delivery, we need a pipeline. This is effectively the pipeline we have been creating in this course.



TARGET ENVIRONMENT

- **Development** – Environment where developers can integrate and test their code
  - Ideally they can do this on their own workstation
  - If not, there needs to be a separate environment

- **Test** – Environment where developers and testers can validate and verify the software

- **Staging** – Environment as close to production as possible. Where acceptance testing and final smoke and deployment tests occur.

- **Production** – Environment where the "live" software is deployed

CONTAINERIZATION

- Containerization is a lightweight alternative to full machine virtualization that involves encapsulating an application in a container with its own operating environment.
- This provides many of the benefits of loading an application onto a virtual machine, as the application can be run on any suitable physical machine without any worries about dependencies.
- The container includes all the dependencies needed to run the application. **Makes it easy to deploy in a consistent manner.**
- Docker is an open-source software product that allows us to define, build and run containers.

DOCKER - BENEFITS

What are some benefits of using Docker?

- Known/standardized configurations – the base image and any additional installed software/configurations are defined in a Dockerfile
- Can start and stop as needed. So can scale up and down as well
- Deployment platforms are built on top of Docker images (i.e., Kubernetes – K8s, OKD)

DOCKER – EXAMPLE FOR A FLASK APPLICATION

```
FROM ubuntu:18.04

LABEL maintainer="youremail@domain.tld"

RUN apt-get update -y && \
    apt-get install -y python-pip python-dev

# We copy just the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /app/requirements.txt

WORKDIR /app

RUN pip install -r requirements.txt

COPY . /app


ENTRYPOINT ["python"]


CMD ["app.py"]
```

Base Image

Image Maintainer

Run Command

Copy Command

Default Dir

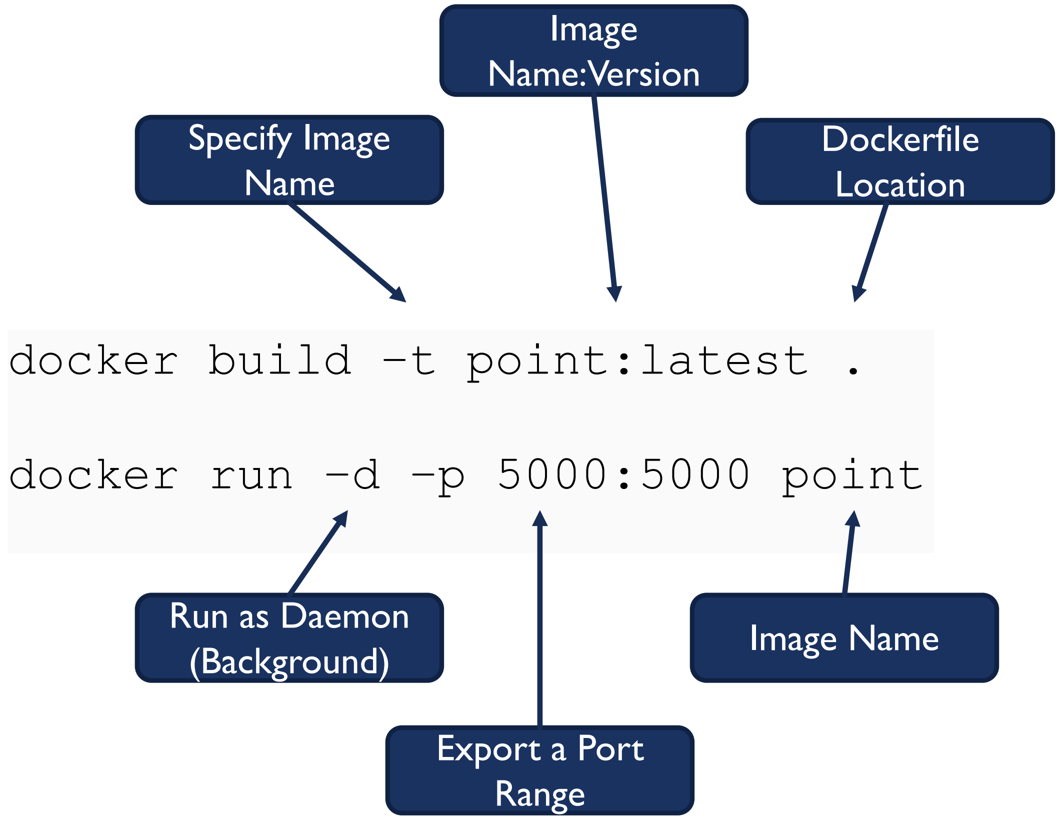
Run Command

Default Dir

Entrypoint – Run when Container Started

CMD – Argument to ENTRYPOINT

DOCKER – EXAMPLE FOR FLASK



```
docker build -t point:latest .

docker run -d -p 5000:5000 point
```

DOCKER – RELEVANT COMMANDS

- docker build – Used to build an image from a Dockerfile
- docker images – Used to list the current set of built images
- docker run – Used to run an image as a container
- docker stop – Used to stop a running image
- docker ps – Used to list the running images

Here is a Docker cheat sheet of commands: https://devhints.io/docker

DOCKER SUMMARY

Docker	Core for building images and running containers.	Use for a small number of independent services.
Docker Compose	Build on top of Docker and part of the Docker Ecosystem.	Use for groupings of related services where the number and type are static (i.e., no scaling). Good for development and test environments, and simple production environments.
Container Orchestration	Typically 3rd party software for deployment of Docker containers, often over a cluster of VMs. Docker does have the built-in "Docker Swarm".	Use for large number of services and/or when services need to be automatically scaled up and down.

NEW LAB REQUIREMENTS

- **(Existing) REQ1180 –** The Enterprise Development Environment shall support an Artifact Management capability. This Artifact Management capability will be Sonatype Nexus 3 AND DockerHub.
- **(Existing) REQ1190 –** The Enterprise Development Environment Continuous Integration capability shall retrieve and publish artifacts from an Artifact Repository.
- **(New) REQ1200** – The Enterprise Development Environment shall use containerization to package applications for deployment to a target environment.
- **(New) REQ1230** – The Enterprise Development Environment shall hide and encrypt any passwords or secrets used in the pipeline builds.

NEW LAB REQUIREMENTS

- **(Existing) REQ1180 –** The Enterprise Development Environment shall support an Artifact Management capability. This Artifact Management capability will be Sonatype Nexus 3 AND DockerHub.

So we will use public repositories on DockerHub as our Artifact Repository for Docker images.

NEW LAB REQUIREMENTS

- **(Existing) REQ1190 –** The Enterprise Development Environment Continuous Integration capability shall retrieve and publish artifacts from an Artifact Repository.

We will publish our Docker images to repositories on DockerHub.

We will have one repository per application for the lab.

NEW LAB REQUIREMENTS

- **(New) REQ1200** – The Enterprise Development Environment shall use containerization to package applications for deployment to a target environment.

We will add a dockerfile for each repository, describing the runtime environment for the application.

We will build a Docker image as part of the build pipeline, and push it to our DockerHub repository.

Next week we will do a simple automated deployment of those containers.

SHARED LIBRARIES – PARAMETERS

- Remember that are shared libraries are Groovy functions:

python_build.groovy

```
def call() {
    <Your Pipeline Here>
}
```

Jenkinsfile

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

SHARED LIBRARIES – PARAMETERS

- We can add parameters to those Groovy functions:

python_build.groovy

```
def call(pipeline_name) {
    ...
    sh "echo ${pipeline_name}"
    ...
}
```

Jenkinsfile

```
@Library('ci_functions@master') _

python_build('Test Pipeline')
```

SHARED LIBRARIES – CONDITIONAL STAGES

- We can make a stage run only when a condition evaluates to True:

```
stage('Package') {
    when {
        expression { env.GIT_BRANCH == 'origin/main' }
    }
    steps {
        ...
    }
}
```

Ref: https://www.jenkins.io/doc/book/pipeline/syntax/#when

We will use this in the next lab week when you exercise a Merge Request (so only pushes to Master builds and deploys a new Docker image.

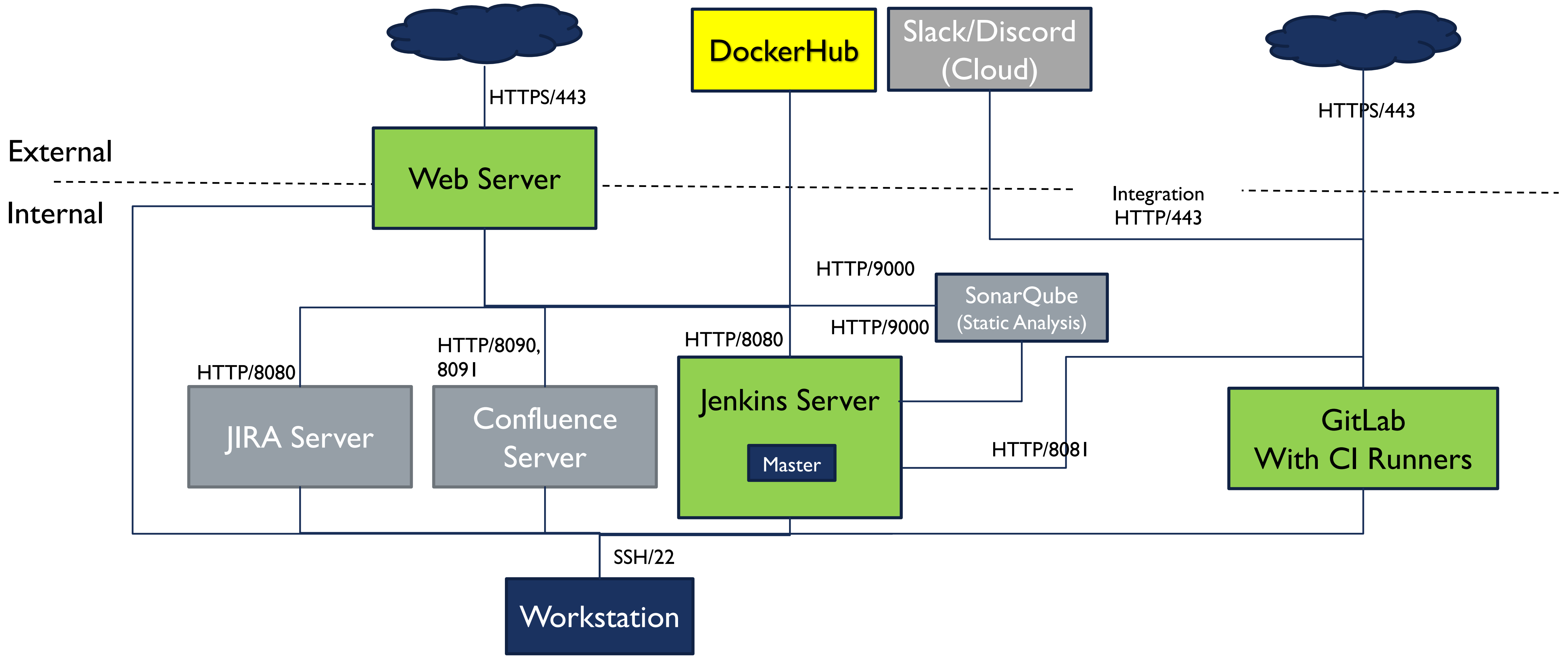
NEW LAB REQUIREMENTS

- **(New) REQ1230** – The Enterprise Development Environment shall hide and encrypt any passwords or secrets used in the pipeline builds.

We will use Jenkins credentials to store the credentials in an encrypted manner and to be able to use the credential without displaying it in the logs.

```
node {
  withCredentials([string(credentialsId: 'mytoken', variable: 'TOKEN')]) {
    sh '''
      set +x
      curl -H "Token: $TOKEN" https://some.api/
    '''
  }
}
```

YOUR ENTERPRISE DEVELOPMENT ENVIRONMENT (SO FAR)



TODAY'S LAB

Lab 9

- Posted to D2L
- Work with your partner
- Demo due by the end of next class