# Open Source Integration Engine Software Configuration Management

Harris SmartWorks
Open Source Integration Engine
Computer Science Dept
COSC 470 / 471
Okanagan College

Aubrey Nickerson
Bao Mai
Derek Manchee
Joseph Egely
Keaton Canuel

April 29, 2021

| Revision Sheet | | | |
|---|---|---|---|
| Revision | Date | Summary of Changes | Name |
| Version 0.1 | 10-30-2020 | Initial Setup | Aubrey Nickerson |
| Version 0.2 | 12-03-2020 | Added Database, Cordova, and hl7 configuration | Aubrey Nickerson |
| Version 0.3 | 04-24-2021 | Added Server Guidelines | Aubrey Nickerson |
| Version 0.4 | 04-26-2021 | Added Bamboo Configuration | Aubrey Nickerson |

# Contents

## 1 Introduction

The goal in this project is to develop an open source integration engine that will take legacy HL7 V2.x messages and send to a cloud based FHIR server. Health Level Seven (HL7) is a set of international standards for transfer of clinical and administrative data between software applications used by various healthcare providers. Hospitals and other healthcare provider organizations typically have many different computer systems used for everything from billing records to patient tracking. All these systems should communicate with each other when they receive new information, or when they wish to retrieve information. HL7 ADT is a messaging standard for communicate Admission, Discharge and Transfer messages.

### 1.1 Purpose

The motivation behind the Software Configuration Plan for the OIE is to give a disentangled review of the product design exercises so those supporting, those performing, and those associating with the product arrangement can acquire an idea of the arrangement. The motivation behind the arrangement, the degree, the meaning of key terms, and references are the four subjects found in this document.

### 1.2 Document Use

This Document is intended to control interior personal on all parts of OIE framework.

## 2 Software

This software section contains configuration information for each piece of software listed

### 2.1 Repository

The Open Source Engine Integration project uses Git for its archive technology. The host program is Bitbucket and is facilitated on the Okanagan school server.

URL: https://bitbucket2020.okanagan.bc.ca/projects/OIE

## 2.2 Version Control

To ideally exploit GIT highlights while likewise considering the setting of the gatherings work style, the accompanying guidelines have been created for fanning.

### 2.2.1 Mainline Branches

There are two branches that are to be stayed up with the latest and erased. These branches are the "master" and "development" branches. Their authorizations have been designed so they can not be changed aside from through a pull request. The users who can consolidate these force demands are Derek Manchee and Aubrey Nickerson.

### 2.2.2 Individual Branches

At the point when an engineer wishes to deal with a specific task, they are to make a branch off of the development branch. The branch will have a naming mapping as follows:

```
ANickerson_OE-23
```

This name comprises of the principal starting of the designer's first name, the engineer's last name, and the number is the JIRA task number. So for Aubrey Nickerson taking a shot at task 23 "ANickerson_OE-23" would be a fitting branch name.

## 2.3 Bamboo Setup

Atlassian Bamboo is a continuous integration and deployment server. It assists software development teams by providing automated building and testing of software source-code status, updates on successful and failed builds, and reporting tools for statistical analysis.

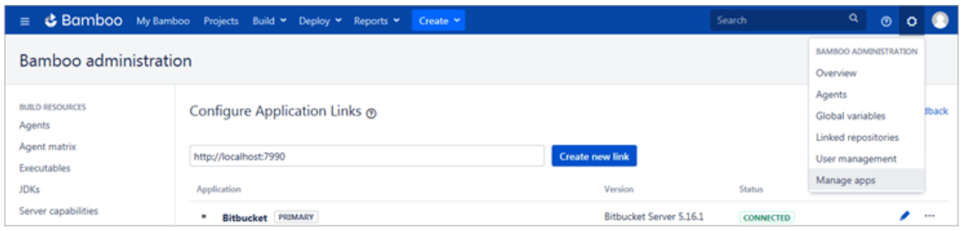
### 2.3.1 Installing and Configuring the Bamboo Server

To download, install and configure the Bamboo server in http://localhost:8085, please look at this tutorial online https://confluence.atlassian.com/bamboo/installing-bamboo-on-linux-289276792.html. The COSC 470/471 team installed Bamboo on Linux CentOS.

### 2.3.2 Configuring Application links with Bitbucket server

As we would be using the source code stored in the Bitbucket repository, we will need to provide and configure the Bitbucket server link in the Bamboo server.

Having logged in to Bamboo URL go to Administration, Manage Apps, Application Links. Add the Bitbucket server URL and click on the Create New Link. This will automatically create a reciprocal link in the Bitbucket server as well.

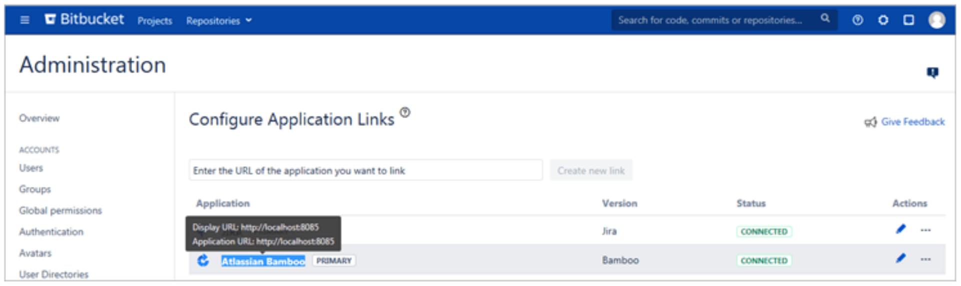


Configure Application Links

http://localhost:7990

| Application | Version | Status |
|---|---|---|
| Bitbucket PRIMARY | Bitbucket Server 5.16.1 | CONNECTED |

Bitbucket Server view of application link is created in the Administration settings.



Configure Application Links

Enter the URL of the application you want to link

| Application | Version | Status | Actions |
|---|---|---|---|
| Display URL: http://localhost:8085 Application URL: http://localhost:8085 | Jira | CONNECTED | |
| Atlassian Bamboo PRIMARY | Bamboo | CONNECTED | |

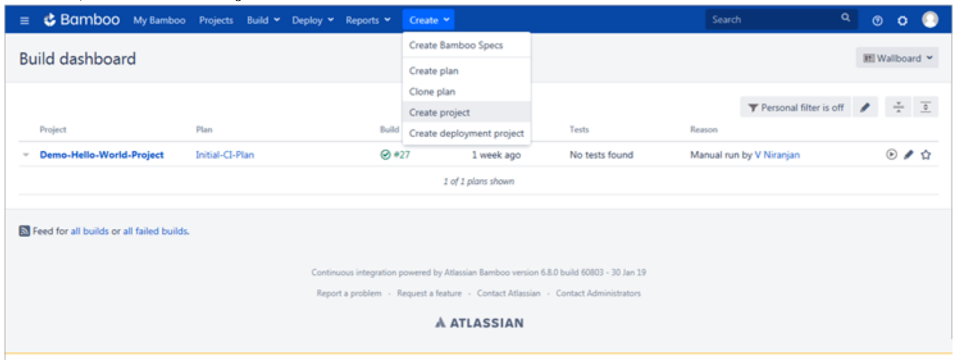
Once the application links are configured, the Bitbucket repositories will be available to be selected in the Bamboo project configuration.

### 2.3.3 Understanding Bamboo Concepts

As you have seen the flow for CI /CD in the previous section, Bamboo uses the concepts of Project, Plan, Stages, and Jobs to accomplish the activities of build and Deploy.

Project: Typically every project teams work on software delivery of multiple applications. The project created in Bamboo is for every application which the team works on.

In order to create a Bamboo project, login to the Bamboo URL and click on Create, Create Project.



Build dashboard

| Project | Plan | Build | | Tests | Reason |
|---|---|---|---|---|---|
| Demo-Hello-World-Project | Initial-CI-Plan | #27 | 1 week ago | No tests found | Manual run by V Niranjan |

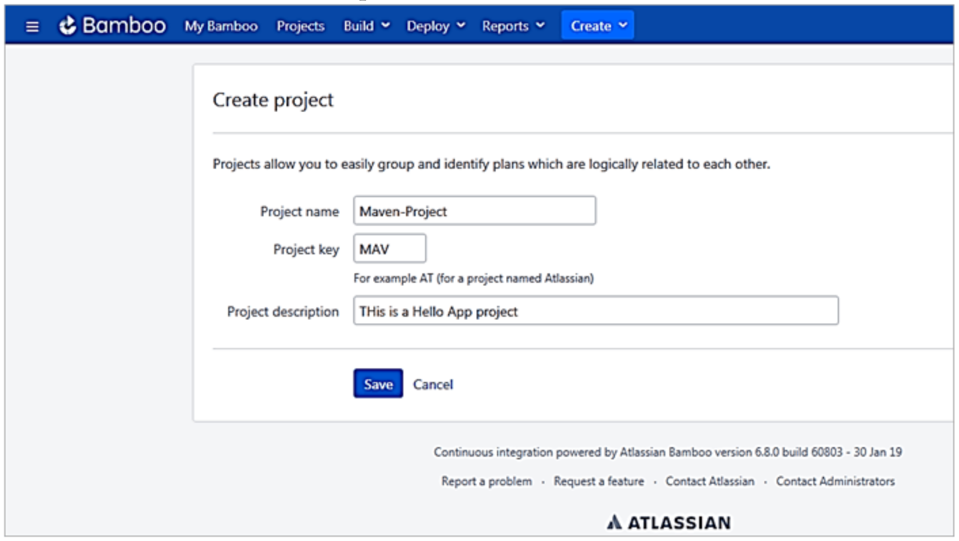
1 of 1 plans shown

Feed for all builds or all failed builds.

Continuous integration powered by Atlassian Bamboo version 6.8.0 build 60803 - 30 Jan 19

Report a problem · Request a feature · Contact Atlassian · Contact Administrators

Enter a name and description. Click on Save.



Create project

Projects allow you to easily group and identify plans which are logically related to each other.

Project name: Maven-Project

Project key: MAV

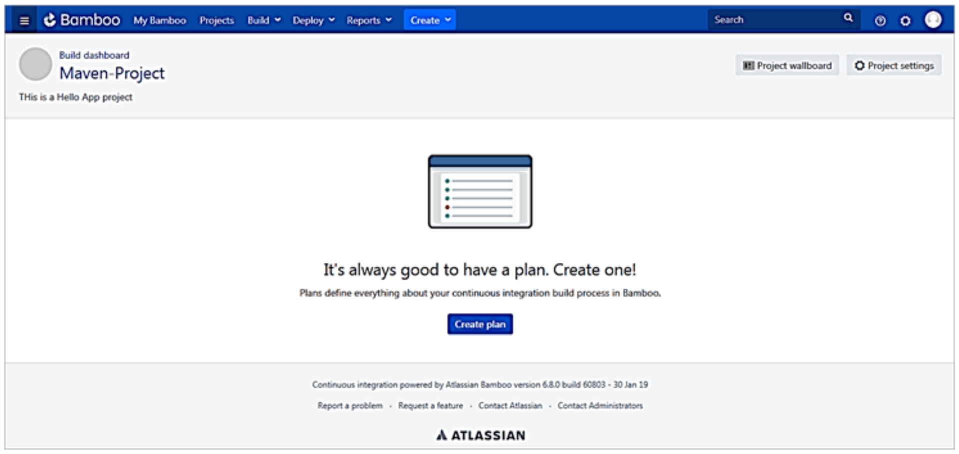
For example AT (for a project named Atlassian)

Project description: THis is a Hello App project

Continuous integration powered by Atlassian Bamboo version 6.8.0 build 60803 - 30 Jan 19

Report a problem · Request a feature · Contact Atlassian · Contact Administrators

Plan: As the project is created, the next step is to create a Plan. A Plan contains information about the version control repository. In this case, our project is stored in the Bitbucket. Few other details like Access control for the plan are also mentioned as a part of creating the Plan.



THis is a Hello App project

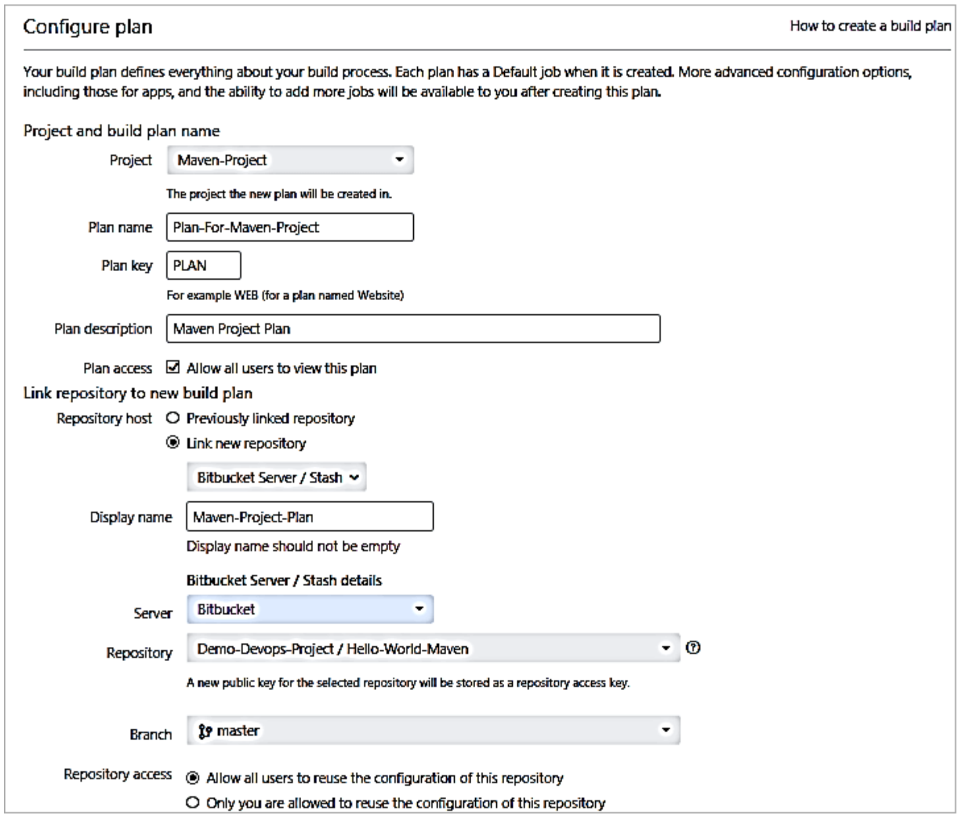
It's always good to have a plan. Create one!

Plans define everything about your continuous integration build process in Bamboo.

Continuous integration powered by Atlassian Bamboo version 6.8.0 build 60803 - 30 Jan 19

Report a problem · Request a feature · Contact Atlassian · Contact Administrators

Click on Create Plan and enter the details as shown below. The Bitbucket repository to be linked to the plan is also available once both the tools are linked.



Configure plan

How to create a build plan

Your build plan defines everything about your build process. Each plan has a Default job when it is created. More advanced configuration options, including those for apps, and the ability to add more jobs will be available to you after creating this plan.

Project and build plan name

Project: Maven-Project

The project the new plan will be created in.

Plan name: Plan-For-Maven-Project

Plan key: PLAN

For example WEB (for a plan named Website)

Plan description: Maven Project Plan

Plan access: [✓ Allow all users to view this plan]

Link repository to new build plan

Repository host: Previously linked repository [✓ Link new repository]

Bitbucket Server / Stash

Display name: Maven-Project-Plan

Display name should not be empty

Bitbucket Server / Stash details

Server: Bitbucket

Repository: Demo-Devops-Project / Hello-World-Maven

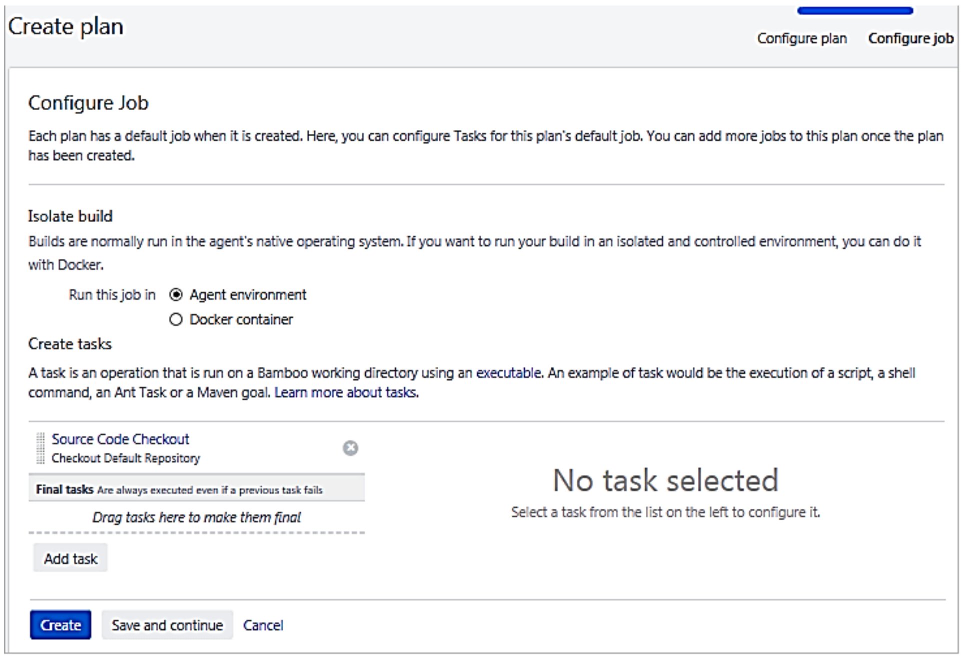
A new public key for the selected repository will be stored as a repository access key.

Branch: master

Repository access: [✓ Allow all users to reuse the configuration of this repository] Only you are allowed to reuse the configuration of this repository

Once the above details are entered, click on Configure plan.

Click on Create for now.



Configure Job

Each plan has a default job when it is created. Here, you can configure Tasks for this plan's default job. You can add more jobs to this plan once the plan has been created.

Isolate build

Builds are normally run in the agent's native operating system. If you want to run your build in an isolated and controlled environment, you can do it with Docker.

Run this job in [✓ Agent environment] Docker container

Create tasks

A task is an operation that is run on a Bamboo working directory using an executable. An example of task would be the execution of a script, a shell command, an Ant Task or a Maven goal. Learn more about tasks.

Source Code Checkout
Checkout Default Repository

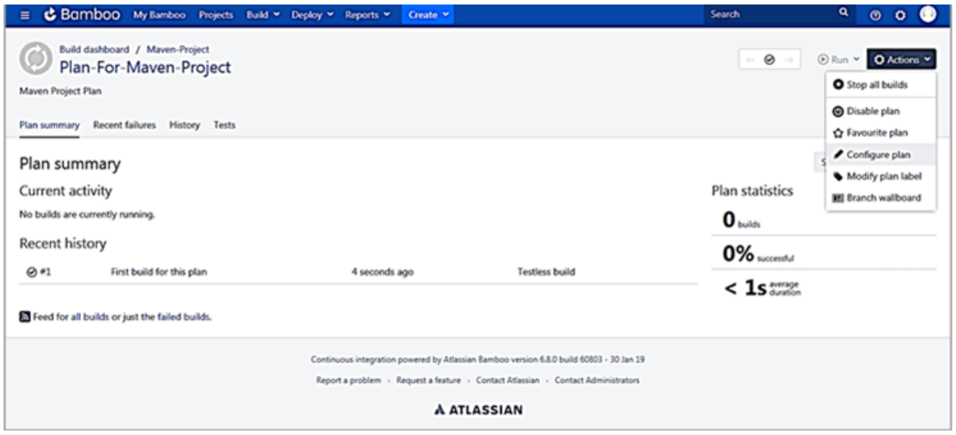
Final tasks Are always executed even if a previous task fails

*Drag tasks here to make them final*

No task selected

Select a task from the list on the left to configure it.

Click on Action then Configure Plan once the plan is created. We will now proceed to create Stages and Tasks.



Plan-For-Maven-Project

Maven Project Plan

Plan summary

Current activity

No builds are currently running.

Recent history

| | | | |
|---|---|---|---|
| #1 | First build for this plan | 4 seconds ago | Testless build |

Feed for all builds or just the failed builds.

Plan statistics

0 builds

0% successful

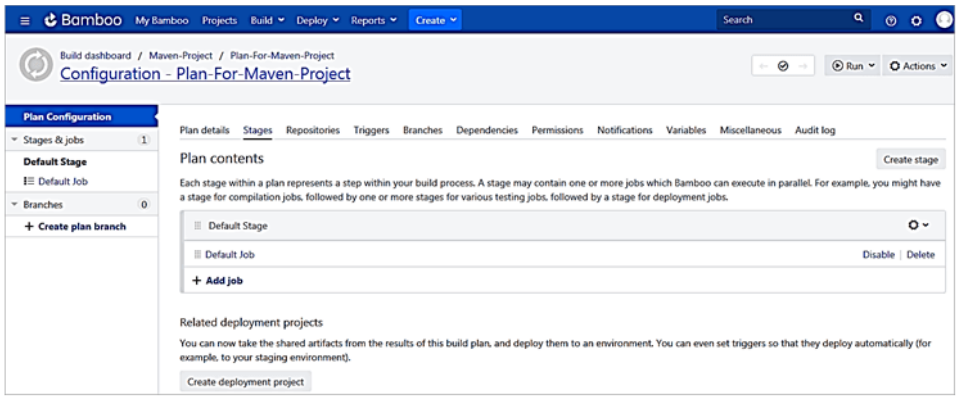
< 1s average duration

Continuous integration powered by Atlassian Bamboo version 6.8.0 build 60803 - 30 Jan 19

Report a problem · Request a feature · Contact Atlassian · Contact Administrators

ATLASSIAN

The following Plan configuration screen that contains a Default Stage in which we will create the Job followed by the grouping of tasks within it comes up.



Configuration - Plan-For-Maven-Project

Plan contents

Each stage within a plan represents a step within your build process. A stage may contain one or more jobs which Bamboo can execute in parallel. For example, you might have a stage for compilation jobs, followed by one or more stages for various testing jobs, followed by a stage for deployment jobs.

Related deployment projects

You can now take the shared artifacts from the results of this build plan, and deploy them to an environment. You can even set triggers so that they deploy automatically (for example, to your staging environment).

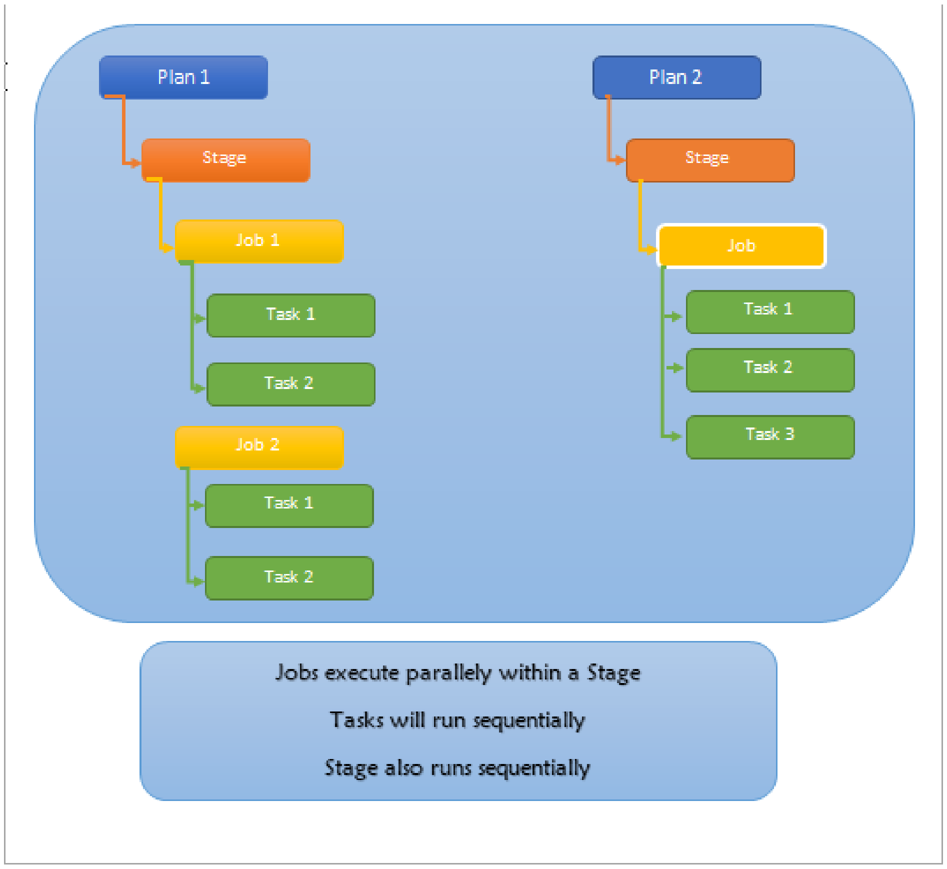
Stage: All the plans created will initially contain a Default stage as shown in the above screen. Example: of a stage can be a Build stage or a Deploy stage. Each stage will contain its own job with the grouping of tasks which is the smallest level of work done for build or deploy.

The stages run sequentially and every stage must execute successfully before moving on to the next stage.

Jobs: Jobs contain one or more tasks which are run in parallel.

Tasks: Tasks are a part of a job. Example: Jobs could be a maven build or deploy to the Tomcat app server.

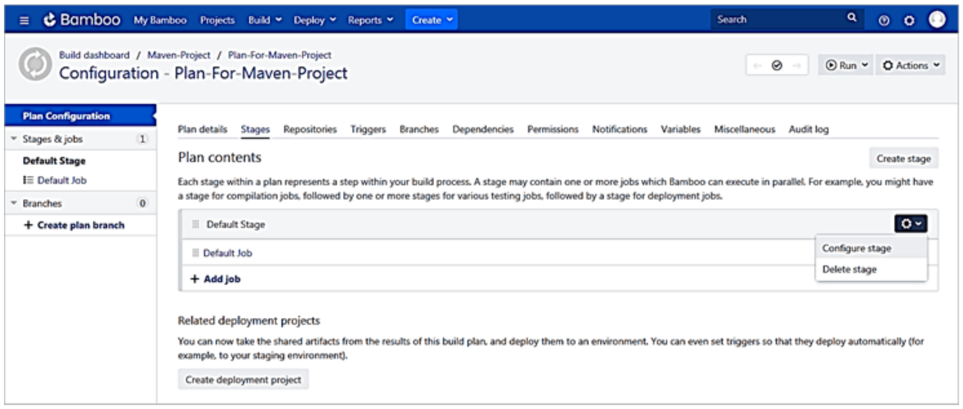
All of the above definitions can be put in the following diagram for execution within a plan and can be used as shown below.


Plan 1
Stage
Job 1
Task 1
Task 2
Job 2
Task 1
Task 2
Plan 2
Stage
Job
Task 1
Task 2
Task 3
Jobs execute parallely within a Stage
Tasks will run sequentially
Stage also runs sequentially

https://www.softwaretestinghelp.com/wp-content/qa/uploads/2019/04/diagram-for-execution-within-a-plan.png

### 2.3.4 Configure Stage

In the Plan Configuration screen, Click on Actions -¿ Configure Stage to rename the definition.



Build dashboard / Maven-Project / Plan-For-Maven-Project

Configuration - Plan-For-Maven-Project

Plan contents

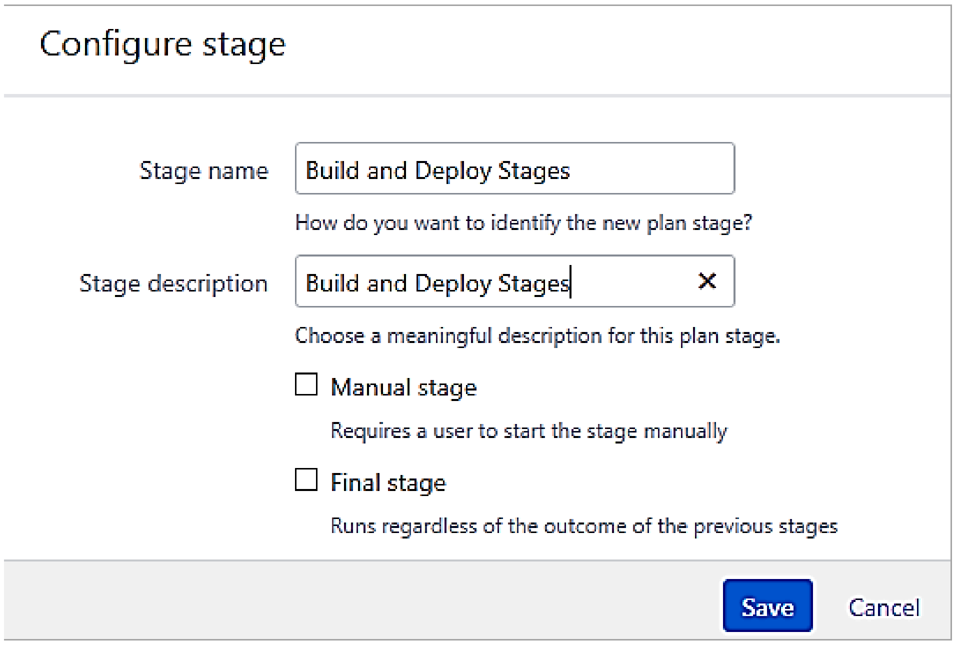
Each stage within a plan represents a step within your build process. A stage may contain one or more jobs which Bamboo can execute in parallel. For example, you might have a stage for compilation jobs, followed by one or more stages for various testing jobs, followed by a stage for deployment jobs.

Default Stage

Default Job

Related deployment projects

You can now take the shared artifacts from the results of this build plan, and deploy them to an environment. You can even set triggers so that they deploy automatically (for example, to your staging environment).



Configure stage

Stage name: Build and Deploy Stages

How do you want to identify the new plan stage?

Stage description: Build and Deploy Stages

Choose a meaningful description for this plan stage.

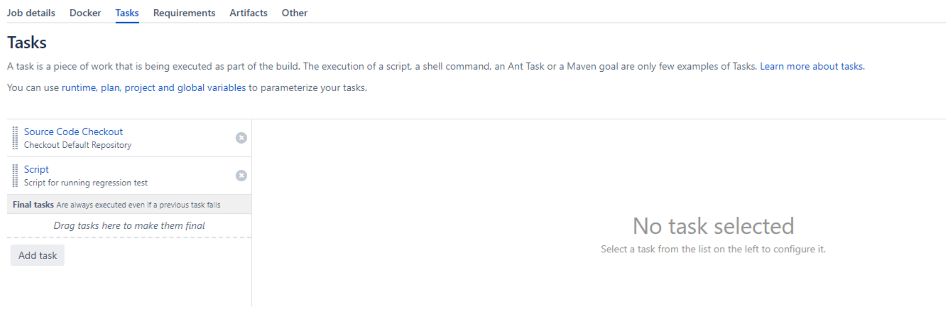
☐ Manual stage

Requires a user to start the stage manually

☐ Final stage

Runs regardless of the outcome of the previous stages

We will now add a task as shown below



Tasks

A task is a piece of work that is being executed as part of the build. The execution of a script, a shell command, an Ant Task or a Maven goal are only few examples of Tasks. Learn more about tasks.

You can use runtime, plan, project and global variables to parameterize your tasks.

Source Code Checkout
Checkout Default Repository

Script
Script for running regression test

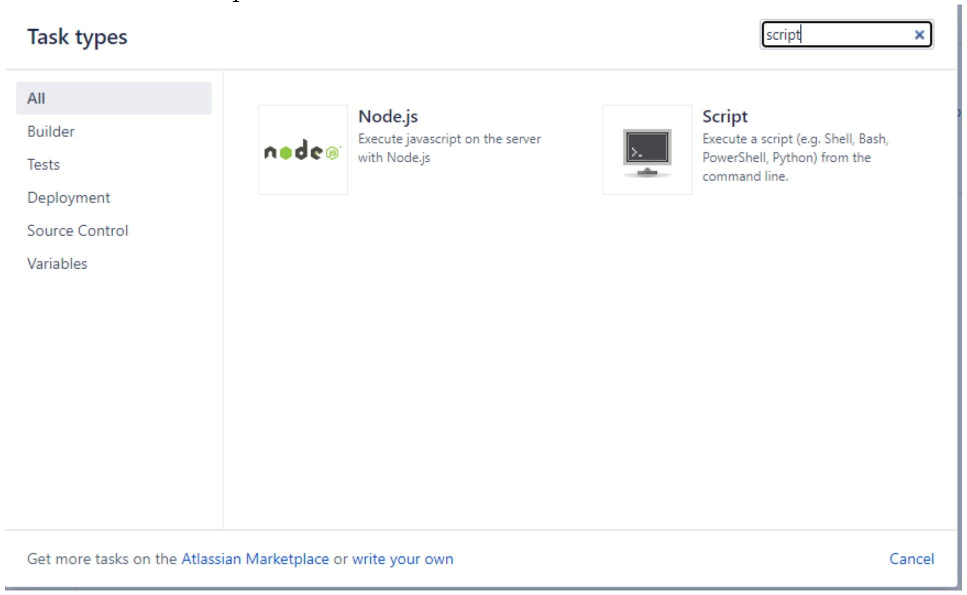
Final tasks Are always executed even if a previous task fails

*Drag tasks here to make them final*

No task selected

Select a task from the list on the left to configure it.

Search for scripts in the search bar and select it.

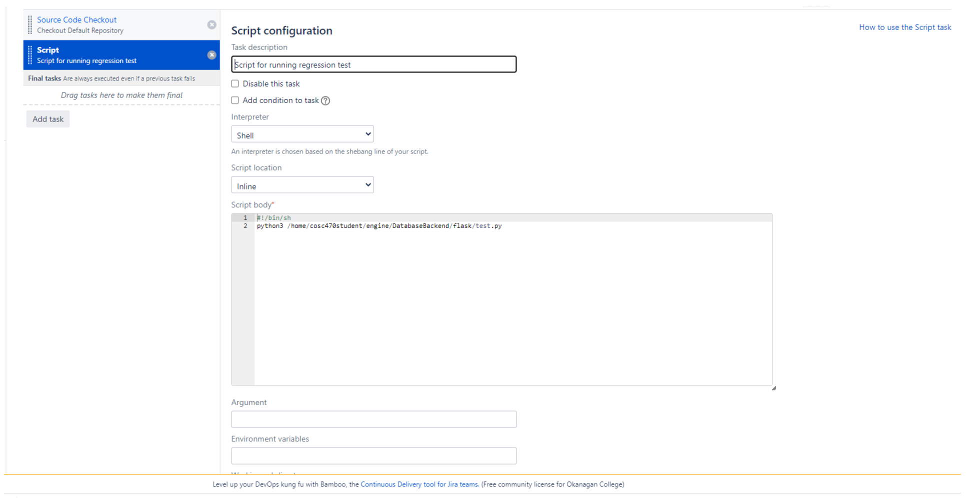


Node.js
Execute javascript on the server with Node.js

Script
Execute a script (e.g. Shell, Bash, PowerShell, Python) from the command line.

Get more tasks on the Atlassian Marketplace or write your own

Fill out the required fields and be sure to create an inline shell script. The shell script will run the unit tests. To run the unit tests, the repository must be cloned on the bamboo server. The shell script looks for the directory that the unit tests are in and will run it when the user manually runs the plan.


Source Code Checkout
Checkout Default Repository

Script
Script for running regression test

Final tasks Are always executed even if a previous task fails

Drag tasks here to make them final

Add task

Script configuration

How to use the Script task

Task description: Script for running regression test

Disable this task

Add condition to task

Interpreter: Shell

An interpreter is chosen based on the shebang line of your script.

Script location: Inline

Script body*

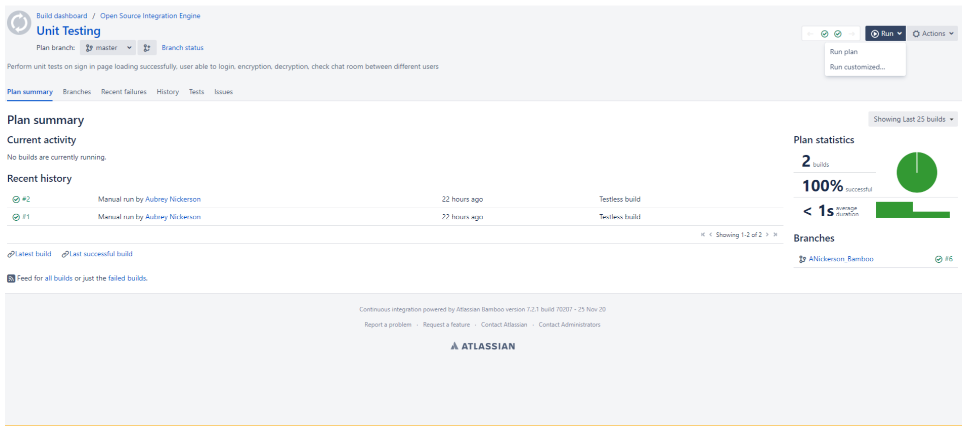
```
#!/bin/sh
python3 /home/cosc470student/engine/DatabaseBackend/flask/test.py
```

Argument

Environment variables

Level up your DevOps kung fu with Bamboo, the Continuous Delivery tool for Jira teams. (Free community license for Okanagan College)

After we set up the task we can run the plan. Click run on the top right corner and click run plan. After running the plan, it should run successfully as shown below.


Build dashboard / Open Source Integration Engine

Unit Testing

Plan branch: master Branch status

Perform unit tests on sign in page loading successfully, user able to login, encryption, decryption, check chat room between different users

Plan summary

Current activity

No builds are currently running.

Recent history

| | | | |
|---|---|---|---|
| #2 | Manual run by Aubrey Nickerson | 22 hours ago | Testless build |
| #1 | Manual run by Aubrey Nickerson | 22 hours ago | Testless build |

Showing 1-2 of 2

Latest build Last successful build

Feed for all builds or just the failed builds.

Plan statistics

2 builds

100% successful

< 1s average duration

Branches

ANickerson_Bamboo #6

Continuous integration powered by Atlassian Bamboo version 7.2.1 build 70207 - 25 Nov 20

Report a problem · Request a feature · Contact Atlassian · Contact Administrators

Bamboo is complex to configure and may take additional research online to get a full grasp.

## 2.4 Continuous Integration

To accomplish legitimate ceaseless reconciliation practices, the accompanying cycle will be trailed by the group.

### 2.4.1 Branching Structure

The structure contains three "levels" of branches which comprises of "master", "development", and individual branches. To accomplish day by day constant joining, the group must union their work upwards.
Ordinary that an engineer works, that developer must make an individual branch for their own work. To do this, they follow these means:

- Pull from development
- Branch from development to create an individual branch that follows the naming conventions documented earlier.

That individual branch may just exist for that day. Toward the finish of that day, the designer must follow these means:

- Add, commit, and push their changes to their individual branch.
- Go to Bitbucket and go to the Pull Request page.
- Create a Pull Request between their individual branch and the development branch.

The pull request will be taken care of by the Product Owner who will check the force solicitation to guarantee that it is adequate. In the event that he regards it such, at that point the Product Owner will acknowledge it and consolidation the progressions into development branch. During this, the Product Owner will likewise close the worker's rendition of that individual branch. On the off chance that the engineer hasn't finished that task, they may reproduce a branch off of improvement with a similar name. They can not keep on essentially utilize a similar branch that they have locally as they are needed to pull from improvement and afterward make another branch off of it.

## 2.5 LatexDiff

To appropriately monitor the contrasts between documentation forms, the device latexdiff will be utilized

### 2.5.1 Installation & Usage

First, one needs to have miktek installed. The download can be found here: https://miktek.org/download

Next, one needs to follow these steps:

- Open Miktek Console
- Click on Packages to open that tab.
- Enter ”latexdiff” as the filter.
- Select the latexdiff package and click the button that looks like a plus sign. This will open the prompt for installation.

### 2.5.2 Usage

From the /miktek/bin/ directory, one can run the latexdiff tool using the command line. The command is as follows:

```
latexdiff vi.tex v2.tex > out.tex
```

The initial two contentions are the documents that are to be looked at. The outcomes should be put in another record that is designated ”out.tex” in this model yet ought to follow a superior naming plan this way:

```
SofwreMgmtDocNov32020
```

As is plainly observed, the configuration is (original document name) and afterward the date. These diff records ought to be kept in a diff organizer in a similar catalog as the report. Diff documents ought to be saved for about around multi week.

## 2.6 Cordova Configuration

Apache Cordova is an open-source mobile development framework. It allows you to use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device’s capabilities such as sensors, data, network status, etc.

### 2.6.1 Installing Cordova

The Cordova command-line tool is distributed as an npm package.

To install the cordova command-line tool, follow these steps:

- Download and install Node.js. On installation you should be able to invoke node and npm on your command line.
- Install the cordova module using npm utility of Node.js. The cordova module will automatically be downloaded by the npm utility.
- $ sudo npm install -g cordova

### 2.6.2 Create the App

Go to the directory where you maintain your source code, and create a cordova project:

```
$ cordova create opensourceintegration com.example.opensourceintegration
OpenSourceIntegrationApp
```

This creates the required directory structure for your cordova app. By default, the cordova create script generates a skeletal web-based application whose home page is the project's www/index.html file.

### 2.6.3 Add Platforms

All subsequent commands need to be run within the project's directory, or any subdirectories:

```
$ cd hello
```

Add the platforms that you want to target your app. We will add the 'ios' and 'android' platform and ensure they get saved to config.xml and package.json:

```
$ cordova platform add ios
$ cordova platform add android
```

#### 2.6.4 Test the App

SDKs for mobile platforms often come bundled with emulators that execute a device image, so that you can launch the app from the home screen and see how it interacts with many platform features. Run a command such as the following to rebuild the app and view it within a specific platform's emulator:

```
$ cordova emulate android
```

Following up with the cordova emulate command refreshes the emulator image to display the latest application, which is now available for launch from the home screen.

## 3 Server Guidelines

The remote server that connects the Open Source Integration app with the back end and database was developed on Linux CentOS 8. It runs on 8 cores and 100 GB of RAM. The back end that handles the data being transferred on the app is called Flask which is a Python web framework used for developing websites. It can be used strictly for back end development and just handle HTTP requests from a client side system which is what the COSC 470/471 team chose to do in this case. The team used a REST API method where the Flask framework takes HTTP requests from the app and snnds data back to the app. The database hosted on the server is a MySQL database. When first loading up Linux CentOS 8, it is best to open the terminal and type:

```
    sudo dnf update
```

### 3.1 Python Configuration

By default RHEL/CentOS 8 doesn't have an unversioned system-wide python command to avoid locking the users to a specific version of Python. Instead, it gives the user a choice to install, configure, and run a specific Python version. The system tools such as yum use an internal Python binary and libraries.

This section will walk the user through installing Python 3 on CentOS 8.

To install Python 3 on CentOS 8 run the following command as root or sudo user in the terminal:

```
sudo dnf install python3
```

To verify the installation, check the Python version by typing:

```
python3 --version
```

At the time of writing this document, the latest version of Python 3 available in the CentOS repositories is "3.6.8":

The command also installs pip.

To run Python, the user needs to explicitly type python3 and to run pip type pip3. In the next section the user will install all of the Python libraries needed to run the back end of the app.

## 3.2 Flask Web Framework Configuration

The first step will be to install all of the pieces that we need from the repositories. We will need to add the EPEL repository, which contains some extra packages, in order to install some of the components we need.

You can enable the EPEl repo by typing:

```
sudo yum install epel-release
```

Once access to the EPEL repository is configured on the system, we can begin installing the packages we need. We will also get a compiler and the Python development files needed by Gunicorn.

```
sudo yum install gcc
```

### 3.2.1 Create a Python Virtual Environment

Next, we'll set up a virtual environment in order to isolate our Flask application from the other Python files on the system.

Start by installing the virtualenv package using pip3:

```
sudo pip3 install virtualenv
```

Now, we can make a parent directory for our Flask project. Move into the directory after you create it. Name the directory osie:

```
mkdir ~/osie
cd ~/osie
```

We can create a virtual environment to store our Flask project's Python requirements by typing:

```
virtualenv osienv
```

This will install a local copy of Python and pip into a directory called osienv within the users project directory.

Before we install applications within the virtual environment, we need to activate it. The user can do so by typing:

```
source osienv/bin/activate
```

The prompt will change to indicate that you are now operating within the virtual environment. It will look something like this

```
(osienv)user@host:~/osie$.
```

### 3.2.2 Set Up a Flask Application

Now that the user is in the virtual environment, we can install Flask and Gunicorn and get started on setting up our application:

Install Flask and Gunicorn

We can use pip3 to install Flask and Gunicorn. Type the following commands to get these two components:

```
sudo pip3 install gunicorn flask
```

### 3.2.3 Install Additional Python Libraries

Lets install all the libraries needed so that the back end of the app can function properly. Below is the command to install all libraries needed. Simply type the command in a terminal.

```
sudo pip3 install eventlet flask_cors flask-socketio
mysql-connector-python cryptography pycrypto
```

We must transfer the backend code from the bitbucket repository into the application folder. The Installation Guide document that was given by the team explains how to clone the repository into the users local computer and gain access of all the source code that builds the app together. This is explained in more detail in pages 17 and 18 of the Installation guide. To transfer those files onto the remote server and into the application folder, the user can use scp. Here is a link demonstrating how to transfer files from server to server using scp. https://linuxize.com/post/how-to-use-scp-command-to-securely-transfer-files/
Follow these directions and change the file names and directories used in the example to the application folder and file names from the repository. So for example, on the users local computer:

- Go to the OpenSource project folder that was created in pages 17 and 18 of the installation guide.
- Open engine folder
- Open DatabaseBackend folder
- Open flask folder
- Right-click and select "Git Bash Here"
- Type

```
scp app.py username@IPAddress:~/osie/
```

The scp command transfered the app.py file to the application folder of the remote server. Repeat this process for dbconfig.py, dbQuerys.py, and PyAES.py. Of course replace the username and IPaddress to the actual username and actual IP associated with the remote server. Transfer these files to the application folder on the remote server using the scp command.

### 3.2.4 Create Flask App

Now that we have Flask available and the back end files transferred to the server, we can create a simple application. Flask is a micro-framework. It does not include many of the tools that more full-featured frameworks might, and exists mainly as a module that the user can import into the projects to assist the user in initializing a web application.

Go to the osie folder on the remote server and type sudo nano app.py. Delete both lines at the very bottom where it says

```
if __name__ == "__main__":
  socketio.run(application, debug=True)
```

Save and exit

Create the WSGI Entry Point
Next, we'll create a file that will serve as the entry point for our application. This will tell our Gunicorn server how to interact with the application.

We will call the file wsgi.py:

```
sudo nano ~/osie/wsgi.py
```

The file is incredibly simple, we can simply import the Flask instance from our application and then run it:

```
from app import application

if __name__ == "__main__":
  socketio.run(application)
```

Save and close the file when you are finished. We will now install MySQL database.

## 3.3 Database Configuration with MySQL

MySQL is an open-source database management system. It implements the relational model and Structured Query Language (SQL) to manage and query data.

Run the following commands to install the mysql.

### 3.3.1 Installing MySQL on CentOS 8

1. install the mysql-server package and a number of its dependencies:

```
$ sudo dnf install mysql-server
```

With that, MySQL is installed on the server but it isn't yet operational. The package installed configures MySQL to run as a systemd service named mysqld.service. In order to use MySQL, it must start with the systemctl command:

```
$ sudo systemctl start mysqld.service
```

To check that the service is running correctly, run the following command. Note that for many systemctl commands — including start and, as shown here, status — we don't need to include .service after the service name:

```
$ sudo systemctl status mysqld
```

If MySQL was successfully started, the output will show that the MySQL service is active:

```
mysqld.service - MySQL 8.0 database server
  Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled;
  Active: active (running) since Thu 2020-03-12 14:07:41 UTC; 1min 7s ago
Main PID: 15723 (mysqld)
  Status: "Server is operational"
   Tasks: 38 (limit: 5056)
  Memory: 474.2M
  CGroup: /system.slice/mysqld.service
          15723 /usr/libexec/mysqld --basedir=/usr
```

Next, set MySQL to start whenever the server boots up with the following command:

```
$ sudo systemctl disable mysqld
```

MySQL is now installed, running, and enabled on the server. Next, we'll go over how to harden the database's security using a shell script that came preinstalled with the MySQL instance.

### 3.3.2 Securing MySQL

MySQL includes a security script that allows to change some default configuration options in order to improve MySQL's security.

To use the security script, run the foillowing command:

```
$ sudo mysql_secure_installation
```

This will go through a series of prompts asking if we want to make certain changes to the MySQL installation's security options. The first prompt will ask whether you'd like to set up the Validate Password Plugin, which you can use to test the strength of your MySQL password.

If we elect to set up the Validate Password Plugin, the script will ask to choose a password validation level. The strongest level — which selected by entering 2 — will require the password to be at least eight characters long and include a mix of uppercase, lowercase, numeric, and special characters:

```
Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD COMPONENT can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD component?

Press y|Y for Yes, any other key for No: Y

There are three levels of password validation policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 2
```

Regardless of whether choosing to set up the Validate Password Plugin, the next prompt will be to set a password for the MySQL root user. Enter and then confirm a secure password for choice:

```
Please set the password for root here.

New password:

Re-enter new password:
```

If you used the Validate Password Plugin, you'll receive feedback on the strength of your new password. Then the script will ask if you want to continue with the password you just entered or if you want to enter a new one. Assuming you're satisfied with the strength of the password you just entered, enter Y to continue the script.

Following that, you can press Y and then ENTER to accept the defaults for all the subsequent questions. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes you have made.

With that, you've installed and secured MySQL on your CentOS 8 server. As a final step, we will test that the database is accessible and working as expected.

### 3.3.3 Testing MySQL

You can verify your installation and get information about it by connecting with the mysqladmin tool, a client that lets you run administrative commands. Use the following command to connect to MySQL as root (-u root), prompt for a password (-p), and return the installation's version:

```
$ mysqladmin -u root -p version
```

If you'd like to connect to MySQL and begin adding data to it, run the following:

```
$ mysql -u root -p
```

Like the previous mysqladmin command, this command includes the -u option, which allows you to specify the user you'd like to connect as (root in this case), and the -p option, which tells the command to prompt you for the user password you set in the previous step.

After you enter your root MySQL user's password, you will see the MySQL prompt:

```
mysql>
```

From there, you can begin using your MySQL installation to create and load databases and start running queries.

### 3.3.4 Enable Remote Connection

In order to connect to the database from our backend, we'll need to enable remote connection. First, you'll need to search for mysql-server.cnf file in your system, this file placement may vary depending on your MySQL version and operating system. The easiest way to find your MySQL configuration file.

On CentOS 8, the mysql configuration file is in:

```
/etc/my.cnf.d/mysql-server.cnf
```

So go to that file and search for the line that begins with bind-address. By default it's set to 127.0.0.1, which means the server will only accept local connections, you need to set it to your external public IP address, or you want to set it to 0.0.0.0 if your IP address isn't static (may change on reboots, etc). If you can't find the line in that file, just add it:

Disable firewall

```
sudo systemctl disable firewalld
```

### 3.3.5 Creating a User for Remote Access

Finally, go to your MySQL server and create a user:

```
mysql> CREATE USER 'python-user'@'%' IDENTIFIED BY 'createpasswordhere';
```

You may be familiar creating users for localhost, in this case, we used '%' character, which means this user can be accessed from any remote host.

We need to grant the newly created user all the privileges, go back to your MySQL server and execute the following command:

```
mysql> GRANT ALL ON *.* TO 'python-user'@'%';
```

With this command, we have granted this user all privileges on all databases and all tables, you can customize that however you want.
Lastly, let's flush the privileges so MySQL will begin using them:

```
mysql> FLUSH PRIVILEGES;
```

### 3.3.6 Create Database

Log onto MySQL as python-user

```
mysql -u python-user -p
```

Create a new database so we can import the tables and data onto the new database.

```
CREATE DATABASE OpenSourceEngDB;
```

Exit MySQL.

### 3.3.7 Import Dump File Into New Database

The SQL backup file that contains all of the tables, patients, doctors, and adminstrations data is located in the repository in the Resources folder. Transfer that SQL file to the server using scp as explained in page 10 and 11 of this document.

Look for the SQL file in the server and go to the directory where its located. Once you change to that directory, you must import the sql file as shown below.

```
mysql -u python-user -p OpenSourceEngDB < OpenSourceEngDB.sql
```

All of the tables and data should now be imported to the new database.

## 3.4 Run Flask Application

Go to the application folder on the remote server.

```
cd ~/osie
```

Open the dbconfig.py file in the application folder on the remote server.

```
sudo nano dbconfig.py
```

And change the credentials to the one you created

```
def connectDB():
mydb = connector.connect(
    host="localhost",
    user="python-user",
    password="password here",
    database="OpenSourceEngDB"
)

return mydb
```

Before moving on, we should check that Gunicorn can run correctly.

We can do this by simply passing it the name of our entry point. We'll also specify the interface and port to bind to so that it will be started on a publicly available interface:

```
    gunicorn --workers=1 --worker-class eventlet --bind 0.0.0.0:8090 wsgi
```

This command will run the web server. If no errors occured then kill the process by CTRL C and type the command again but adding –daemon at the end. For example:

```
gunicorn --workers=1 --worker-class eventlet --bind 0.0.0.0:8090 wsgi --daemon
```

This command will run the web server in the background and will run forever unless interrupted. To stop it from running in the background, type:

```
    pkill gunicorn
```

All HTTP requests can now be received on this server.

If you followed all of the instructions on the Installation Guide, you should now be able to run the app as a whole and be able to login as a user, accessing the login credentials from the database, and accessing the additional features of the app.

## 4 HL7 Configuration

python-hl7 is a simple library for parsing messages of Health Level 7 (HL7) version 2.x into Python objects. python-hl7 includes a simple client that can send HL7 messages to a Minimal Lower Level Protocol (MLLP) server (mllp_send).

python-hl7 currently only parses HL7 version 2.x messages into an easy to access data structure. The library could eventually also contain the ability to create HL7 v2.x messages.

python-hl7 parses HL7 into a series of wrapped hl7.Container objects. The there are specific subclasses of hl7.Container depending on the part of the HL7 message. The hl7.Container message itself is a subclass of a Python list, thus we can easily access the HL7 message as an n-dimensional list. Specifically, the subclasses of hl7.Container, in order, are hl7.Message, hl7.Segment, hl7.Field, hl7.Repetition. and hl7.Component.

python-hl7 includes experimental asyncio-based HL7 MLLP support in MLLP using asyncio, which aims to replace txHL7.

### 4.1 Install hl7

python-hl7 is available on PyPi via pip or easy_install:

```
$ sudo pip3 install -U hl7
```

For recent versions of Debian and Ubuntu, the python-hl7 package is available:

```
$ sudo apt-get install python-hl7
```

### 4.2 Run HL7 Listener

The HL7 listener and sender are located in the repository in the TestScripts folder. It is straightforward to run. The hl7 sender sends a message to the listener and the listener sends the message to the database. The HL7 listener includes an additional file that sorts the messages called message_sorting.py In this file, the program takes the message from the listener and sorts the data into a JSON file. The JSON file is stored onto the users computer and it sends the JSON file to the remote servers database.

```
async def output_To_Json(dic):
    directoryPath = "C:/Path/To/JSON"
    jsonFile= time.strftime("%d%m%Y%Hh%Mm%S") + ".json"
    fullJSONPath = directoryPath + jsonFile

    with open(fullJSONPath, 'w') as fp:
        json.dump(dic, fp, indent=4)
        url = "http://remote_ip_address:8090/sendFile"
        files = [('document', (fullJSONPath, open(fullJSONPath, 'rb'),
        'application/octet')),]
        fp = requests.post(url, files=files)
        fp.close()
```

To run the listener all you have to do is open a terminal and type

```
python3 testHL7Receiver.py
```

While the listener is running, open the testHL7Sender.py and run.

```
python3 testHL7Sender.py
```

The message will be sent to the receiver and the receiver will pass the message to the message_sorting.py file so that the messages can be sorted and sent to the database.

## 5 Conclusion

This document explains how to use the software tools that were used during the project and how the server is configured to run the back end logic of the app. The document also includes how to run the hl7 listener.