**CI/CD (Continuous Integration/Continuous Delivery)** a method to deliver apps to customers by introducing automation into the stages of app development. **Continuous integration** is the process of merging changes to the main branch of a repository as often as possible. These changes are then validated by creating a build and running automated tests against the build. **Continuous delivery** is an extension of continuous integration since it automatically deploys all code changes to testing and/or production environments after the build stage.

In this project, I created a CI/CD pipeline to deploy a Flask application. The goal of this project is creating the pipeline according to specific instructions, meanwhile identifying ways to improve or further automate.

The tools I used in this project:

* Git
* GitHub
* Jenkins
* AWS EC2
* AWS Elastic Beanstalk

**Step1: Code Development**

* In GitHub, fork/copy a repository containing application and config files to GitHub profile.

Graphical user interface, text

Description automatically generated

* Create a local git repository (on ubuntu virtual machine) then connect my local repository to remote repository (GitHub)

Command to create a git repo:

Git init

Commands to connect local git rep to GitHub:

ssh-keygen -t rsa -b 4096 -C "[myemail@mail.com](mailto:myemail@mail.com)"

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id\_rsa

(cat id\_rsa.pub then Add the ssh public key to GitHub)

git remote add origin [git@github.com:SterlingMcKinley/kuralabs\_deployment\_1.git](mailto:git@github.com:SterlingMcKinley/kuralabs_deployment_1.git)

Command to test connection from local to GitHub:

ssh -T git@github.com

Text

Description automatically generated

* Cloned a repository from GitHub ( <https://github.com/SterlingMcKinley/kuralabs_deployment_1.git>) to my local git repository

Text

Description automatically generated

(The above steps allow version control for any changes made to the application and/or configuration files.)

Now that I have pulled the original version of the application, I can either modify or test the application. I choose to test the application to determine functionality or if any updates are required. In order to locally test the application, I must create a virtual environment. Virtual environments are beneficial to prevent configuration drift.

* + The following command will create and activate my virtual environment (be sure the current location is the local git repo)
    - sudo apt install pip #Pip a tool for installing Python packages
    - pip - -version #Checking version of pip
    - pip install virtualenv pip #Tool that creates virtual env for Python app
    - vim .bashrc

(-Add the below line to the BOTTOM of .bashrc file)

PATH=$PATH:/home/<your username>/.local/bin #Sets env variable

* + - pip list #Lists installed packages
    - virtualenv venv #Creates virtual environment
    - export FLASK\_APP=application #exports the variable for the application
    - source ./venv/bin/activate #activates the virtual environment
    - pip install Flask #install Flask package
    - flask run #start the flask app within virtual environment

Text

Description automatically generated

That the virtual environment is running allowing the flask app to be served locally, you must attempt to access the application. The url to the application will be:

localhost:5000 or <http://127.0.0.1:5000>

Graphical user interface, application

Description automatically generated

THE APPLICATION WORKS! Now I can integrate the application to the next phase as well as another tool to perform additional tests.

**Step2: Continuous Integration**

* In order to merge any changes into the test and automation process, I must build the integration environment. I will be building an EC2 instance. On that instance I will install Jenkins.
* To create an EC2 instance we must complete a couple step that I will explain below.
  + Go to AWS console, sign in and navigate to the EC2 service/console

<https://console.aws.amazon.com/ec2/>

* + On the left panel, navigate to Network & Security then select Key Pairs
    - Create a key pair with a uniquely identifiable name.
    - Download the key pair and save it locally.
      * Permissions must be closed/private. (chmod 400 <keyfile>.pem)

Graphical user interface, text, application

Description automatically generated

* + On the left panel, navigate to Network & Security then select Security Groups
    - Click Create Security Group
      * Enter an identifiable name for the security group
      * On the Inbound section, rules must be configured.
        + Add 3 rules and the following configs

Type: SSH | Source: Custom & in the drop-down box, select 0.0.0.0/0

Type: Custom TCP Rule | Port: 8080 | Source: Custom & in the drop-down box, select 0.0.0.0/0

Type: HTTP | Port: 80 | Source: Custom & in the drop-down box, select 0.0.0.0/0

* + - * + Lastly, select Create

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated

The pre-steps for creating the EC2 instance is complete.

* + - Go to EC2 Dashboard, then select Launch Instance and
      * Name the instance ( ie ubuntu\_jenkins)
      * Application and OS Images: Ubuntu
      * Instance type: (in this example t2.micro – Free tier eligible)
      * Key pair: (select the key pair previously created)
      * Network settings: select existing security group then choose the security group we previously created
      * Select Launch Instance

(EC2 instance will take a few minutes to complete. On completion, the instance will be running)

Graphical user interface, text, application

Description automatically generated

Lets connect to the EC2 instance. Installing Jenkins will be next.

* Connect to EC2 instance (Ubuntu Linux)
  + Navigate to EC2 > Instances and Select the “instance name”
  + The lower half of the console will display an instance summary.
    - Copy the Public IPv4 DNS

Graphical user interface, text, application, email

Description automatically generated

* + Go to your terminal or ssh client to ssh into the EC2 using port 22
    - Execute ssh command to connect to the EC2 instance

ssh -i </the/location/of/keypair>.pem [ubuntu@ec2-XX-XXX-XX-XXX.compute-1.amazonaws.com](mailto:ubuntu@ec2-XX-XXX-XX-XXX.compute-1.amazonaws.com)

Text

Description automatically generated

* Installing Jenkins in the EC2 instance
  + Execute the following commands on the fresh EC2 Ubuntu Linux instance

#installs java runtime engine for java apps like Jenkins

sudo apt update && sudo apt install default-jre

#Adds the repository key to your system/EC2 instance

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg

#Append the Debian package repository address the EC2 Ubuntu Linux source.list

sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

#updates packages and dependencies

sudo apt update

#install Jenkins and its dependencies

sudo apt install jenkins

* + Start Jenkins

sudo systemctl start jenkins.service

#command to check if the Jenkins services is active/running

sudo systemctl status jenkins

Text

Description automatically generated

* + Go to your browser to use the Jenkins GUI
    - The Jenkins GUI URL is the Public IPv4 IP address of the EC2 instance using port 8080

<http://XXX.XX.XXX.XXX:8080>Graphical user interface, application

Description automatically generated

* + Create user credentials and login
  + In order to connect Github to Jenkins an access token must be created.
    - In GitHub, navigate to GitHub settings > developer settings > SSH and GPG keys
    - Select personal access token and create a new token
    - Create a name for the token and select the below settings for access token permissions:
      * Repo
      * Admin:repo\_hook
    - Select Generate token

A screenshot of a computer

Description automatically generated with medium confidence

* + Lets prepare a multibranch build
    - Select “New item”
    - Select multibranch pipeline then select ok
    - Enter a display name & brief description
      * Display Name: Build Flask
      * Description: CI/CD pipeline deployment 1
    - Under Branch Sources, select Add Source and select GitHub

Graphical user interface, text, application

Description automatically generated

The Jenkins console window will change

* + - Select Jenkins

Graphical user interface, text, application, email

Description automatically generated

* + - In the middle of the page,
      * Username: enter your GitHub username
      * Password: enter the token
      * Then select save

Graphical user interface, text, application, email

Description automatically generated

* + - Under Repository HTTPS URL
      * Provide the GitHub repository URL and press validate
        + You will see output to confirm the repo URL

Graphical user interface, text, application, email

Description automatically generated

* + - IMPORTANT! PLEASE BE SURE BUILD CONFIGURATION is consistent with the below screenshot

Application

Description automatically generated with low confidence

* + - Select Apply then select Save
    - Immediately, a build will be triggered. If no build is in progress, select “Scan Repository”

Graphical user interface, application, Word

Description automatically generated

If the build fails, the result will show. In this case, I test the application locally using a virtual environment to ensure the application was functional.

Jenkins is used to build and test software applications. This process makes tasks easier for developers to integrate or update projects. Jenkins automating task will save time also increase business productivity.

**Step2: Continuous Delivery**