

Deployment 2

September 27, 2022

By: Kingman Tam- Kura Labs Cohort 3

Purpose:

To further our understanding of deploying applications and creating a CI/CD pipeline that automates tasks.

After having completed Deployment 1, a solid understanding of the deployment pipeline was established. A program is compiled into a repository that includes:

- README file- A file describing the application and other important information
- application.py script- The main script that will run the program and call different files from the repo
- Pipfiles and a requirement.txt file- Used to create an environment that the program was designed for
- Jenkinsfile- A guide for Jenkins that lists stages of testing (building/testing/etc)
- test.py file- A file that Jenkins uses for its testing stage
- Other files that the program needs to reference in order to run

This repository was linked to Jenkins which was then built and tested according to the Jenkinsfile. If it passed the test, the repository was compressed and manually uploaded to an Elastic Beanstalk server which created the environment that the program would be hosted on.

As all of the steps taken required manual input, speed and efficiency of the deployment was hindered. Deployment 2 addresses some of these issues and introduces the concept of Continuous Deployment and Continuous Integration.

Steps:

1. Launch an EC2 with Jenkins installed and activated
 - Jenkins is the main tool used in this deployment for pulling the program from the repository, then building, testing, and deploying it to a server.
 - Steps for installing Jenkins from deployment 1 were followed:
 - wget Jenkins keyrings
 - Added Jenkins repo to sources.list.d with keyrings
 - Installed Java, pip, and python3.10-venv
 - Start Jenkins in terminal and set up in Jenkins GUI
2. Create a Jenkins user in AWS account
 - A new IAM user named “EB-user” was created that has its own access keys and AdministratorAccess permissions
 - This user is used by AWS-CLI in later steps
3. Install AWS CLI on the EC2
 - The Amazon Web Service Command Line Interface is installed so that the user can run AWS commands through the Linux Terminal
 - The ‘curl’ command was used to get the file “awscliv2.zip” from the Amazon AWS server

- The file was decompressed with “unzip” and then the ‘install’ file ran
4. Configure AWS CLI
 - In order to be able to use the AWS CLI, an IAM user must be linked to the terminal.
 - In the jenkins user (su jenkins), “aws configure” was run and the access keys from the EB-user (step 2) were entered along with the region and output format (json).
 5. Install Elastic Beanstalk on jenkins EC2 user
 - AWS Elastic Beanstalk CLI is installed on the jenkins user so that the application can be deployed to elastic beanstalk from the terminal
 - “pip install awsebcli --upgrade --user” is run so that elastic beanstalk is installed only on the user that ran the command (jenkins)
 - eb is installed onto the users .local/bin folder which is not on the default PATH. PATH must be updated to include /var/lib/jenkins/.local/bin. See **troubleshooting** section below.
 6. Connect GitHub to Jenkins Server
 - Jenkins needs to be able to scan a repository from GitHub and use the contents of the repo to build and test the environment that the application will run on.
 - A token for the repo is created in GitHub and input into the Jenkins application through the Jenkins GUI when Jenkins creates a new pipeline for the project
 - If successfully linked, Jenkins should automatically start building from the GitHub repo
 7. Deploy application from EB CLI
 - After Jenkins follows the step as stated in the Jenkinsfile, a workspace is created.
 - “eb init” is run in the workspace directory of the jenkins user and the user is prompted with configurations. Workspace path can be found in Jenkins when in the build screen. See **troubleshooting** section below.

The screenshot displays the Jenkins web interface. At the top, the Jenkins logo is on the left, and a search bar, a notification icon, the user name 'Kingman Tam', and a 'log out' button are on the right. Below the navigation bar, a breadcrumb trail reads 'Dashboard > Build Flask > main > #1 > Workspaces'. The main heading is 'Workspaces for Build Flask » main #1'. Underneath, a single workspace is listed: '• /var/lib/jenkins/workspace/ur1-shortener_main on built-in'. A left-hand sidebar contains various icons and labels for build actions: Status, Changes, Console Output, Edit Build Information, Delete build '#1', Git Build Data, Test Result, Restart from Stage, Replay, Pipeline Steps, and Workspaces. The 'Workspaces' item is highlighted. At the bottom right of the page, the text 'Jenkins 2.361.1' is visible.

- “eb create” is run in the same directory and elastic beanstalk should launch the application from the terminal
- This step can be verified in AWS by navigating to Elastic Beanstalk and clicking on the environment that was created.

8. Add Deployment stage to the pipeline in Jenkinsfile

- Thus far, Jenkins has been connected to the GitHub repo containing the application. Whenever an update or change is made on the files of the repository, one would have to log into Jenkins, build the new commits, and then return to the terminal to manually launch the updated environment.
- By adding a deployment stage, Jenkins can now run the command:
`sh '/var/lib/jenkins/.local/bin/eb deploy {{Your environment name}}'`
 through its workspace directory which will “Deploy the application source bundle from the initialized project directory to the running application.” (source: [AWS](#))
- In other words, after Jenkins successfully “Builds” and “Tests” the new repo, it will automatically “Deploy” it (as long as there is a running environment).

9. Modify the pipeline

- The events in the pipeline are dictated by the Jenkinsfile. Any commands added to this file will be executed in its appropriate stage.
 - Another test file was created and uploaded to the GitHub repository. Any file in the repo with a name that starts with “test_” or ends with “_test” will be read by pytest. Within those files, a function needs to be defined with a name starting with “test_” to be run.
 - “test_urls.py” was created to test for the content in the “urls.json” file in the repository. The test was run by Jenkins and results were posted as follows:



Console Output

```
===== test session starts =====
platform linux -- Python 3.10.6, pytest-7.1.2, pluggy-1.0.0 -- /var/lib/jenkins/workspace/url-shortener_main/test3/bin/python3
cachedir: .pytest_cache
rootdir: /var/lib/jenkins/workspace/url-shortener_main
collecting ... collected 2 items

test_app.py::test_home_page PASSED [ 50%]
test_urls.py::test_url PASSED [100%]

- generated xml file: /var/lib/jenkins/workspace/url-shortener_main/test-reports/results.xml -
===== 2 passed in 0.16s =====
```

- Notifications were also added to the pipeline.
- In order for Jenkins to utilize different tools, plugins must be used. For this notification, the “Slack Notifications” plugin needed to be installed.
- After the plugin is installed, the “slackSend” command can be used in the Jenkinsfile script.
- The Jenkins file was modified to include “slackSend” messages in the post stages as per [jenkins pipeline documentation](#) and [Jenkins-Slack documentation](#).

Kingman Tam 12:30 PM
 added an integration to this channel: [jenkins](#)

jenkins APP 12:39 PM
 Slack/Jenkins plugin: you're all set on <http://44.202.55.38:8080/>

jenkins APP 3:48 PM
 FYI: jenkins-url-shortener-main-9 has SUCCESSFULLY completed its 'BUILD' stage
 FYI: jenkins-url-shortener-main-9 has SUCCESSFULLY completed its 'TEST' stage
 ATTENTION: jenkins-url-shortener-main-9 has FAILED its 'DEPLOY' stage

New

- Note: all changes and documents can be viewed on [GitHub](#)

Continuous Integration:

Continuous Integration involves “a developers ability to modify and integrate new code throughout the development cycle” (source: [IBM](#)). During this deployment, (not included in the steps above) A webhook was created that allowed GitHub to push the repository to Jenkins whenever commits were made to the main branch. When modifying the pipeline (step 9), every time an update was made and committed to the main branch (test file added or Jenkinsfile modified), Jenkins would automatically start going through the stages to build and test. This is an important function of CI. The ability to add functions in branches and have them built and tested automatically at any point of development is fundamental to Continuous Integration.

Continuous Deployment:

On the other hand Continuous Deployment “is a strategy in software development where code changes to an application are released automatically into the production environment” (source: [IBM](#)). Adding this step to Deployment 2 was a key difference from Deployment 1. Instead of manually compressing and uploading the application files to an elastic beanstalk environment each time the code changed, the deploy stage (with the command “eb deploy”) was added to the Jenkinsfile so that the updated code would be automatically deployed to the existing environment after the build and test stages were completed. The changes would take effect immediately without needing further manual input.

Issues/Troubleshooting:

- The last command we were asked to run on page 1 of the instructions, was to ‘su’ into the jenkins user. The next commands to run in the terminal on page 5 asked us to curl, unzip, and sudo install the AWS CLI. The jenkins user is unable to run these commands as it does not have sudo privileges (unzip needed to be installed with apt). This caused confusion as to what commands were supposed to be run as the ubuntu user and which were supposed to be run as the jenkins user. I ultimately decided that all of the commands up until “aws configure” on page 6 would be done by the ubuntu user.
- After installing awsebcli, a warning was printed to the terminal saying “WARNING: The scripts futurize and pasteurize are installed in '/var/lib/jenkins/.local/bin' which is not on PATH.”. Using the command “eb --version” returned “eb: command not found”. I navigated to ~/.local/bin and saw that the eb command was there.

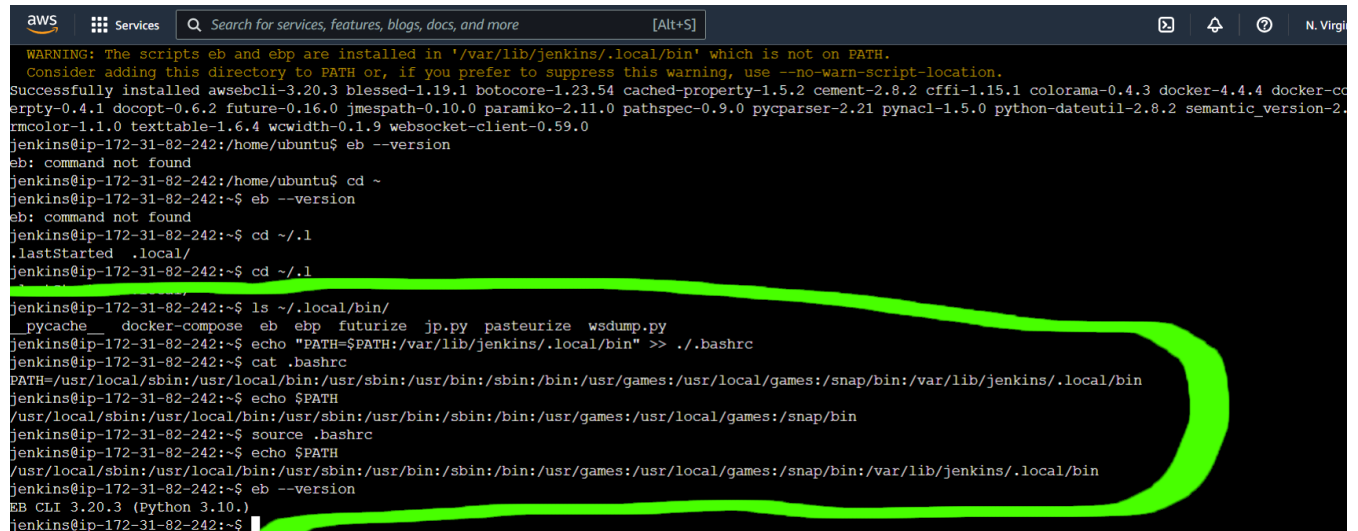
```

aws
Services
Search for services, features, blogs, docs, and more [Alt+S]
N. Virginia Kingman Tam

Building wheel for docopt (setup.py) ... done
Created wheel for docopt: filename=docopt-0.6.2-py2.py3-none-any.whl size=13723 sha256=c3c88d03f39658d3113a1f3fe31c46cef5f1e2a9f86062f0c8a2c6a97935c25f
Stored in directory: /var/lib/jenkins/.cache/pip/wheels/fc/ab/d4/5da2067ac95b36618c629a5f93f809425700506f72c9732fac
Successfully built awsebcli
Installing collected packages: wcwidth, texttable, termcolor, docopt, awsebcli, semantic-version, pycparser, pathspec, jmespath, colorama, websockets, python-dateutil, dockerpty, cffi, blessed, pynacl, docker, botocore, paramiko, docker-compose, awscli
WARNING: The scripts futurize and pasteurize are installed in '/var/lib/jenkins/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The script docker-compose is installed in '/var/lib/jenkins/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The scripts eb and ebp are installed in '/var/lib/jenkins/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed awsebcli-3.20.3 blessed-1.19.1 botocore-1.23.54 cached-property-1.5.2 cement-2.8.2 cffi-1.15.1 colorama-0.4.3 docker-4.4.4 docker-compose-1.25.5 docopt-0.6.2 dockerpty-0.6.3 future-0.16.0 jmespath-0.10.0 paramiko-2.7.1 pycparser-2.21 pynacl-1.5.0 python-dateutil-2.8.2 semantic-version-2.8.5 six-1.14.0 te
jenkins@ip-172-31-82-242:/home/ubuntu$ eb --version
eb: command not found
jenkins@ip-172-31-82-242:~$ eb --version
eb: command not found
jenkins@ip-172-31-82-242:~$ cd ~/.local/bin/
jenkins@ip-172-31-82-242:~/.local/bin$ ls
awscli  eb  ebp  futurize  jp.py  pasteurize  wsdump.py
jenkins@ip-172-31-82-242:~$

```

With all of this information, I knew that `/var/lib/jenkins/.local/bin` needed to be added to the PATH environment variable. Usually, a `.bashrc` file is updated with the PATH but the jenkins user did not have such a file. I was considering ‘exporting’ the PATH variable in the terminal but knew that it was temporary and would revert back after logging out of the session. After a long search, I found that the PATH that jenkins user uses is echoed from the file `/etc/environment`. I modified that path, rebooted the EC2, and the PATH was updated in the jenkins user as well. This took 2 attempts. The first attempt led me to overwrite the PATH in a way where none of my commands worked until I exported the PATH on the terminal to reset what I did. The second attempt worked fine but I figured that modifying that file is probably not the best method to complete this task. After brainstorming with some colleagues, we realized that manually creating a `.bashrc` file in the jenkins home and updating the PATH there would run `.bashrc` whenever the user is loaded and thus update the PATH variable.



```

aws Services Search for services, features, blogs, docs, and more [Alt+S] N. Virgil

WARNING: The scripts eb and ebp are installed in '/var/lib/jenkins/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed awscli-3.20.3 blessed-1.19.1 botocore-1.23.54 cached-property-1.5.2 cement-2.8.2 cffi-1.15.1 colorama-0.4.3 docker-4.4.4 docker-c
erty-0.4.1 docopt-0.6.2 future-0.16.0 jmespath-0.10.0 paramiko-2.11.0 pathspec-0.9.0 pycparser-2.21 pynacl-1.5.0 python-dateutil-2.8.2 semantic_version-2.
rmcolor-1.1.0 texttable-1.6.4 wcwidth-0.1.9 websocket-client-0.59.0
jenkins@ip-172-31-82-242:/home/ubuntu$ eb --version
eb: command not found
jenkins@ip-172-31-82-242:/home/ubuntu$ cd ~
jenkins@ip-172-31-82-242:~$ eb --version
eb: command not found
jenkins@ip-172-31-82-242:~$ cd ~/.l
.lastStarted .local/
jenkins@ip-172-31-82-242:~$ cd ~/.l
jenkins@ip-172-31-82-242:~$ ls ~/.local/bin/
_pycache_ docker-compose eb ebp futurize jp.py pasteurize wsdump.py
jenkins@ip-172-31-82-242:~$ echo "PATH=$PATH:/var/lib/jenkins/.local/bin" >> .bashrc
jenkins@ip-172-31-82-242:~$ cat .bashrc
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/var/lib/jenkins/.local/bin
jenkins@ip-172-31-82-242:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
jenkins@ip-172-31-82-242:~$ source .bashrc
jenkins@ip-172-31-82-242:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/var/lib/jenkins/.local/bin
jenkins@ip-172-31-82-242:~$ eb --version
EB CLI 3.20.3 (Python 3.10.1)
jenkins@ip-172-31-82-242:~$

```

- On page 18 of the instructions, we are asked to `cd` into `"/var/workspace/{{The name of your project}}/"`. This directory does not exist and was not created by Jenkins. Instead, the workspace that Jenkins uses was located in the “home” of the jenkins user at `/var/lib/jenkins/workspace`. Navigating to the workspace through the Jenkins GUI revealed the correct path to the directory.
- Adding an additional test to the pipeline was difficult for me due to the fact that I am not so familiar with python or the functions that are used in creating an application. I was able to create a test by parsing through the json data in the file `“urls.json”`. This test is not a true test of the functionality of the application. It is more a test of whether the `urls.json` file exists in the repository and if it was modified in a specific manner.
- Adding notifications to the pipeline was also difficult in that the documentation for the email/slack plugins were either outdated or for a different version of Jenkins. The email plugin required that I enter both my username and password to use the function which I did not feel was secure. The slack plugin called “Global Slack Notifier” which was supposed to notify via Slack the completion of jobs did not show up in the configuration page. Instead, the “Slack Notification” plugin was used which required that the `Jenkinsfile` be modified so that it includes a `“post”` section after the build stages in order to send messages.