Continuous Build & Delivery Pipeline

George McNally

1.Introduction to CI/CD

The high pressures put on companies to continuously deliver software and applications can put a burden on development teams, and if not handled correctly can deteriorate a team's performance and overall capabilities. This is where CI/CD comes into effect. CI/CD is a mechanism put in place to speed up development and delivery. It enforces an element of automation into the development lifecycle, and so, frees up a lot of the developer's time to focus on the code. CI/CD mainly focuses on continuous integration, delivery, and deployment. It connects each part of the development lifecycle, from integration to testing to delivery and deployment. It is usually adopted by a development team implementing an agile philosophy. [1]

Continuous integration (CI) involves developers frequently pushing code to a repository whenever a change is made. The repository is integrated with a tool that pulls the code from it, creates a war or jar file, and performs automated tests to ensure there have been no mistakes, and the code base is free from errors or bugs. The main role for continuous integration is to implement a strategy whereby the building, packaging and testing of applications are done in an automated manner. This leads to a well-oiled machine, where developers are consistently committing code changes, therefore improving the quality of the application as well as the overall teamwork.[3]

Continuous delivery and deployment (CD) comes after the integration stage. The delivery stage involves the automation of delivering a finished product to a testing stage. If the code passes the tests, it is then deployed and pushed into production. This means deployment of an application is as easy as pressing a button, enabling the ability to release software at whatever rate the developer chooses, be it weekly, daily or monthly. Continuous integration is enforced while the code is still being written, whereas continuous delivery is pushed through once the code has been completed. [1]

Implementing CI/CD comes with a wealth of benefits such as increased speed of delivery, shortened development cycle, reduction in bugs post-development, increased release frequency, and reduced complexity of dev lifecycle. Dzone notes that in recent years there has been a major jump in small-to-medium businesses turning to CI/CD. [2] Those who have implemented CI/CD have automation as a part of the testing and deployment. It was also noted that many companies have turned to CI/CDaas (CD-as-a-service) as companies are open to outsourcing some of the pipeline duties to third-parties.

Although there are many benefits, CI/CD also comes with certain technical difficulties. Dzone notes that 47% of companies implementing CI/CD struggle with maintaining legacy infrastructures.[2] It has also been noted that transitioning out of old infrastructures and

processes can be a challenge. When it comes to organizational challenges, developers have noted their struggles when dealing with legal and regulatory constraints. 45% also struggle with aligning pipelines and processes across multiple teams.[2] This should not deter anyone from looking into CI/CD as the benefits seem to outweigh the negatives.

When it comes to implementing CI/CD, developers have numerous tools to choose from, one however is used way more frequently than the rest. That tool is Jenkins. A survey held by Dzone found that 71% of developers implementing CI/CD used Jenkins, this was followed by GitlLab at 32%, and then Azure pipelines at 14%.[2]

Leading CI/CD Platforms/Tools				
Jenkins	GitLab	Azure Pipelines	AWS CodePipeline	Bamboo
71%	32%	14%	10%	9%

Diagram i. Leading CI/CD Platforms/Tools [2]

When implementing CI/CD it is common practice to share a single source of version control. It should hold all artifacts. Popular version control repositories include Github, Docker, and more. It is also essential that the testing strategy has a strong foundation. To ensure this, teams should work off a test pyramid with between 3-5 layers of testing, including software and infrastructure testing. There are many tools when it comes to automated testing such as Junit for unit tests, Selenium and Cucumber for performance tests and penetration test tools for security testing. Sonarqube is another testing tool which analyses code quality, and reports if there are any bugs or defects.

There are many pipeline design patterns, two of which are “fast team feedback”, and “trigger the right pipeline”. For “fast team feedback”, every time a developer commits a piece of code it triggers the right pipeline through the use of automation. The build pipelines are programmed for speed and urgent notification of issues. [4] The “trigger the right pipeline” design pattern enables the ability for each branch commit, pull, or merge to trigger a different pipeline behaviour, depending on the developer’s needs. These are just two examples, but many more can be implemented.[4]

With the help of CI/CD tools and design patterns, CI/CD can help ease the workload of a team and enable the chance to work on other parts of the project if needed. Being notified of bugs and defects at an early stage cuts the cost of repair, so overall the resources needed for a project are shortened. [5]

2.User Story & Application Architecture

Architecture

The application is a microservice and is held on a tomcat container, it represents a shop-service and is linked to a h2 in memory database and implements a full set of CRUD

operations. Built with the help of maven, and on the Spring framework, the user can access the application, and its database, through a browser or a rest client such as postman. The microservice is made up of a model, a REST controller and a repository.

Pipeline User Story

As a developer, I want to automate the development and deployment cycle of my application so I can detect bugs and defects in my code while also speeding up the development process.

Pipeline Acceptance Criteria

System must be automated from the moment code is pushed to repository.
This must include testing and deployment.

Application User Story 1

As a user I want to search for a shop by name so I don't have to scroll through a list to find it

Application Acceptance Criteria 1

I can search for shop by name.
I cannot see a shop with a different name
I can see all of the matching shop's information.

Application User Story 2

As a user I want to search for shops by country so I can see all shops in my country.

Application Acceptance Criteria 2

I can search for shops by country.
I can only see shops from specified country, no other shops can be seen.
I can see information for all shops from specified country.

3. Pipeline Description & Stages

As seen in the diagram below, the pipeline used for this project included the building, testing, and deploying of an application. It helps throughout the development cycle as once a push has happened to the repository a pipeline is triggered which builds, tests and deploys the application to a tomcat container.

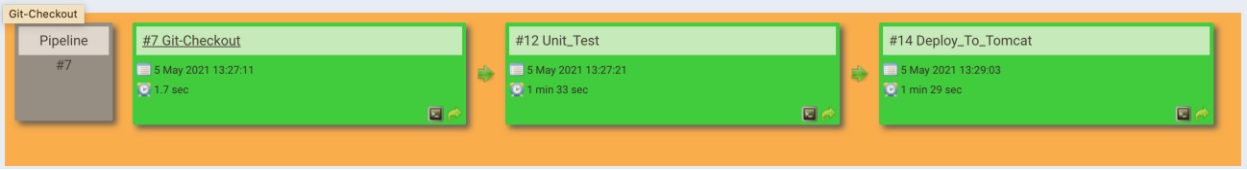

Diagram iii. Pipeline.

Stage 1 - Git_Checkout

As this stage utilizes the "Poll SCM" build trigger, it checks the application's git repository every minute and if there are any changes it activates a build. It holds a simple command of echo 'download code' which returns that string if the console can connect to the git repository. As seen in the diagram below, the console then uses a post build action to trigger the build of a downstream project and the next stage of the pipeline which is the Unit_test stage.

```
+ echo 'download code'
download code
Triggering a new build of Unit_Test
Finished: SUCCESS
```

Diagram iii. Git_checkout console message.

Stage 2 – Unit_Test

Once the Git_checkout is successful, a unit_test build is triggered. The unit_test is configured to connect to the git repository, look for the application's pom.xml file, do a clean of the application and the run whatever tests are in the test folder. It will then return the results of the tests, let the user know if the build was a success and will trigger a build of the deploy_to_tomcat stage. Overall, there were ten tests taken, 7 of which were testing the applications controller, 2 tested the shop class methods, and 1 did a basic test on the shop application class. Junit was used to undertake the tests as well as mockito. Sonarqube was also used to test the code quality and to find any defects.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[JENKINS] Recording test results
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  44.677 s
[INFO] Finished at: 2021-05-05T13:28:13+01:00
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /Users/george/.jenkins/workspace/Unit_Test/pom.xml to com.ait.store/shop-service/0.0.1-
SNAPSHOT/shop-service-0.0.1-SNAPSHOT.pom
channel stopped
Triggering a new build of Deploy_To_Tomcat
Finished: SUCCESS
```

Stage 3 – Deploy_To_Tomcat

Once the testing stage is successful, the deployment to tomcat is triggered. Deploy_to_Tomcat is configured to connect to the application's git repository, it then does a clean install package, which forces Maven in to cleaning the service before running the install which makes sure to clear any compile files in the system and compiles the service from scratch. It then packages the application in a war file and is configured to deploy that war file to a tomcat container. The result of this can be seen below.

```
[INFO] Processing war project
[INFO] Building war: /Users/george/.jenkins/workspace/Deploy_To_Tomcat/target/shop-service-0.0.1-SNAPSHOT.war
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.4:repackage (repackage) @ shop-service ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  54.308 s
[INFO] Finished at: 2021-05-05T13:30:07+01:00
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /Users/george/.jenkins/workspace/Deploy_To_Tomcat/pom.xml to com.ait.store/shop-service/0.0.1-
SNAPSHOT/shop-service-0.0.1-SNAPSHOT.pom
[JENKINS] Archiving /Users/george/.jenkins/workspace/Deploy_To_Tomcat/target/shop-service-0.0.1-SNAPSHOT.war to
com.ait.store/shop-service/0.0.1-SNAPSHOT/shop-service-0.0.1-SNAPSHOT.war
channel stopped
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /Users/george/.jenkins/workspace/Deploy_To_Tomcat/target/shop-service-0.0.1-
SNAPSHOT.war to container Tomcat 8.x Remote with context shop-service
  Redeploying [/Users/george/.jenkins/workspace/Deploy_To_Tomcat/target/shop-service-0.0.1-SNAPSHOT.war]
  Undeploying [/Users/george/.jenkins/workspace/Deploy_To_Tomcat/target/shop-service-0.0.1-SNAPSHOT.war]
  Deploying [/Users/george/.jenkins/workspace/Deploy_To_Tomcat/target/shop-service-0.0.1-SNAPSHOT.war]
Finished: SUCCESS
```

Once this has been done, the application is deployed and can be accessed through the specified port number. Eg, localhost:8083/shops.

4. Pipeline Evaluation

This pipeline works as expected. It is fully automated from the moment a piece of code is pushed to a repository. That includes building the application, testing the application for bugs, and if the tests pass, deploying the application to a Tomcat container. This cuts out many steps you would have to do manually and speeds up the development and deployment process.

There were, however, a few processes cut out of the originally planned project due to the lack of access to AWS. Given the chance, Docker would have been implemented as a way to simplify the process that goes into managing containers. On top of this, Docker is portable and is a way to split the application from the infrastructure in order to produce software in a fast manner. Given that Docker would be included, both Antifactory and Ansible would also be added to the pipeline. Artifactory is a docker registry, and is used to manage docker images while also making use of access control. With Artifactory, it possible to manage docker images from the

development stage straight through the pipeline and into deployment and distribution. Ansible is used for task automation and orchestration. It can set tasks and implement a number of events to take place across numerous servers and/or devices.

Overall, the implemented pipeline worked well and cut a lot of development and deployment work once it was set up. Not having to run tests manually saves a lot of time and having the application deployed to tomcat each time a push commit was triggered made the procedure a lot easier.

3. References

[1] Devops. What is CI/CD. Devops. Viewed 01/05/2021. https://www.redhat.com/en/topics/devops/what-is-ci-cd

[2] Garee, R. (2020). The State of CI/CI - Trend Report. View 02/05/2021. https://dzone.com/trendreports/the-state-of-cicd

[3] Pittet, S. (). Continuous integration vs. continuous delivery vs. continuous deployment. Viewed 02/05/2021. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[4] Shriver, R. (2020). 7 Pipeline Design Patterns for Continuous Delivery. Viewed 2/05/2021. https://www.singlestoneconsulting.com/blog/7-pipeline-design-patterns-for-continuous-delivery/

[5] Brown, T. (2020) CICD Best Practices. Viewed 02/05/2021. https://opensource.com/article/20/5/cicd-best-practices