POLITECNICO DI TORINO

Master's Degree in Engineering And Management

Master's Degree Thesis

Adopting Continuous Integration, Continuous Delivery, Continuous Deployment, and Continuous Testing in DevOps

Supervisor:
Prof. Demagistris Paolo Eugenio

Candidate:
Olapada Faith Tosin

Academic Year 2020/2021

Abstract

Nowadays, Software companies are required to deliver features and products frequently, faster, and seamlessly.
To accomplish these, the software delivery pipeline should be agile and must diverge from traditional waterfall practices. This needed shift from traditional software development practices to a presumably better approach birthed a new practice called DevOps. DevOps is a set of practices breaching the gap between development (Dev) and IT operations (Ops). Derived from agile practices, DevOps looks to shorten the software development life cycle providing continuous delivery and testing and ultimately, ensure higher software quality.

This thesis will shed light on DevOps practices, experiment with building different types of CI/CD pipelines, evaluate the different stages in a DevOps pipeline with a strong focus on the cloud leaders Amazon web services (AWS).

Finally, key metrics achieved with DevOps practices will be compared to metrics derived from traditional software practices.

Table of Contents

List of Figures

1 Introduction

1.1 What is DevOps?

From the Waterfall model, organizations moved to the Agile model, DevOps is an evolution of the agile model, although not a replacement. DevOps is a set of practices that integrates software development (Dev) and operation (Ops). DevOps includes tools, cultural and technical practices that accelerates software and feature delivery in an organization, compared to the rate of software release using traditional software development and IT operations processes. This increase in delivery time helps the organization battle competition more effectively while simultaneously increasing customers satisfaction.

With the advent of the internet came disruptions of various industries, from the financial industry to the music industry, to ecommerce. A commonality with these disruptions has come in the role software plays in the value chain, software does not just support business activities, it has become an important part of every organization. In most case, customers interact with products and services offered by a company through a website or applications on different devices. Different software are also used to improve operational efficiency by innovations made in parts of the value chain, such as communication, operations and logistics. To be able to compete effectively, and meet the constant changing standards, organizations must improve their software delivery pipelines [1].

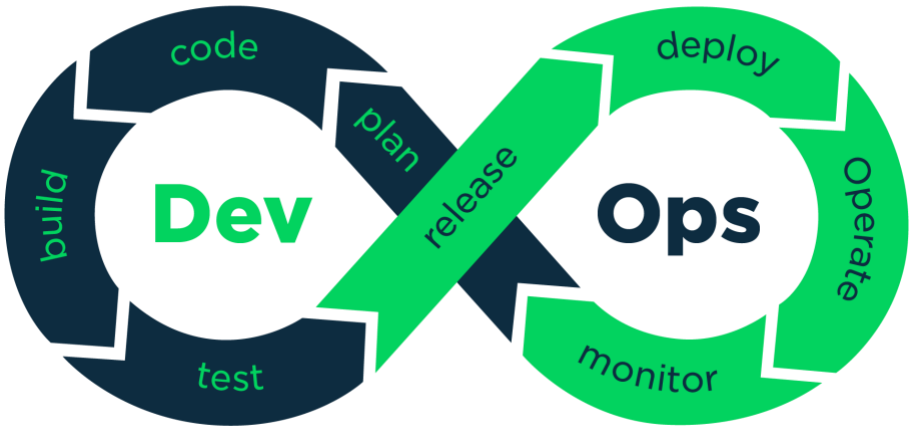

Figure 1.1:Stages of a DevOps Pipeline

With DevOps practice, the development and operations teams work very closely. Either by a merger into a single DevOps team or linked by a specialized DevOps engineering team. This team works across the entire application lifecycle, from

development, build, test and application deployment to operations including monitoring. Seldomly, DevOps teams might include quality assurance (QA) and security engineers. The goal of the DevOps team is to automate manual or repetitive processes in the application lifecycle, using sets of tools.

DevOps tools are tools used in assisting cross departmental human collaboration and interaction, enabling continuous delivery, continuous build, continuous testing, continuous monitoring, and maintaining overall software reliability [2].

Adoption of DevOps requires a new type of mindset and organizational culture. To achieve the best results, there must be a complete removal of any barriers between the development and the IT operations team, otherwise separated in a traditional model. Communication and coordination between these teams should be seamless.

1.2 DevOps Practices

To innovate and ship software faster using the DevOps methodology, there are some important practices which should be adopted by an organization.

1.2.1 Continuous Integration

Continuous integration is a software development practice where assigned actions in the central repository like developers merging changes to a specified branch, triggers the automated software build and test. With continuous integration, there is a cultural practice of integrating code changes as frequently as possible and an automation side (using CI and build tools like Jenkins or AWS CodePipeline). The main aim of this stage is to improve software quality by reducing software delivery time and early bug detection [1].

In a CI stage of the software delivery pipeline, developers frequently commit, merge or push code to the central repository (version control-based system like GitHub) after testing in a local environment. After the pipeline is triggered, a build of the pushed code is done, after which unit testing is carried out to further verify the built software is error free.

Traditionally, developers would work independently for a long time before merging their codes for build. This process meant, bugs are accumulated over this period and features are delivered to the customers much slower. Using CI, bugs are found and addressed quicker, developer's productivity is increased with frequent code merge and customers get features and updates faster.

1.2.2 Continuous Delivery

Continuous delivery is another software development practice where successful code builds are made ready for the production environment i.e. for customer usage. Continuous delivery further automates the process of the software pipeline, it improves upon continuous integration by deploying the built code changes to a specified deployment environment. Depending on the organization practice and it size, deployment environment may include test environments (dev or pre-prod) or production environment after a successful build. This means, after a developer pushes code changes to a specified branch, the code is built, and unit tested automatically (continuous integration) and the artifacts from the stage is deployed straight to a specified environment (dev, pre-prod or prod).

With innovations in cloud computing, this stage has become cheaper and easier to implement, compared to practicing on premises. At this stage, developers can schedule and automate more types of tests on the now deployed and functional deployment.

There is often a confusion with continuous delivery and continuous deployment stages of the pipeline, especially when abbreviated as CD (could mean continuous deployment or continuous delivery). With continuous delivery, deployment to the production environment needs a manual approval while with continuous deployment, deployments or software release is automated from end to end without a manual stage. The use of either continuous delivery or deployment will largely depend on organization maturity and needs.

1.2.3 Continuous Deployment

As with continuous delivery, continuous deployment is a more matured software development practice. Here, deployment requires no manual intervention or stage, end to end deployments are fully automated. As with continuous delivery, developers can addition tests like UI testing, stress testing, integration testing etc. Continuous deployment also makes use of another DevOps concept: Infrastructure as a code (IaaC) or Infrastructure as code (IaC), which allows organizations or companies to store and version control their infrastructure as they do application code. Infrastructural code can either live with the respective applications in the same repository or versioned separately in its own repository.

1.2.4 Continuous Testing

Continuous testing refers to automating all test cases or scenarios. From unit testing, load testing, UI testing, integration testing to API reliability testing. The declared test case should be run at their respective stages i.e., unit testing during the CI stage, load testing at CD stage. Continuous testing reduces errors in the final software by detecting bugs way earlier than normal. Taking time to write test cases is not a required stage to a achieving a fully automated CI/CD pipeline but, definitely a best practice and a must for large organizations.

1.2.5 Continuous Monitoring

After software deployment is complete, it important to actively monitor but application performance and its infrastructure. Monitoring application logs provides organization with greater insight on different metrics, like customer experience with each deployment, data collected can then be used as feedback for future deployments. Monitoring infrastructural performance can also help the DevOps team make improvements in performance optimization and optimizing future environments.

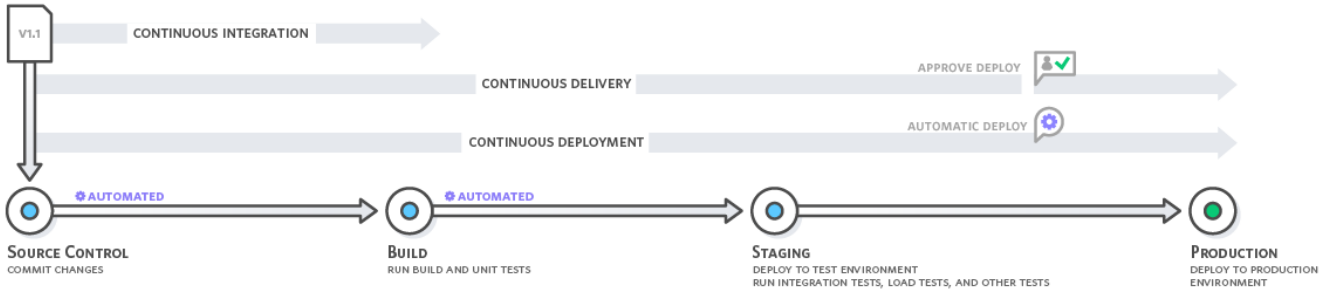

Figure 1.2:DevOps Practices

1.3 DevOps Tools

There are tools used in each stage of the CI/CD pipeline, to help automate the software delivery process. Practicing DevOps relies on a heavy usage of automation tools, from multi-team collaboration tools, code version control tools, continuous integration tools, continuous testing tools to observability or active monitoring tools. The importance of proper tooling in DevOps cannot be over emphasized but, DevOps is more than tools.

Figure 1.3:Periodic Table of DevOps Too

1.3.1 Collaboration Tools

The cultural aspects of DevOps entails there is inter-departmental human collaboration. Through the use of chat applications, tickets or issue tracking systems and wikis, information sharing, and consistent communication should be adopted. In most case, source control tools like GitHub or Gitlab offers issue tracking and wiki creation supports.

Tools like Microsoft teams and Slack also fall into this category. They promote collaboration by offering instant call and messaging services to teams and organizations.
Confluence by Atlassian, built to improve collaboration and knowledge sharing across teams. Confluence not only enables creation of documentations and product requirements, but it also integrates with other Atlassian suite of products like Jira and Trello for extended project management functionalities.
The mentioned collaboration tools and others play an important role in speeding up communication and collaboration across teams (development team, operations team, marketing and sales team), leading to a better alignment of project goals and increased success rate.

1.3.2 Source Code Management Tools

Source control tools help developers in a software development teams collaborate and work more effectively. Development using a source control tool has become a basic requirement in software development, developers can work on separate features or branches and merge to make a fully functional product. Actions in the source control is set to trigger the CI/CD pipeline.

The most common versioning tools are Git and SVN. Although products like GitHub, Gitlab and Bitbucket provide a Git hosting repository for developers with baked in tools to ship better code.

Source code management tools are the first point of integration for automating software release. The pipeline can be set to start build on every successful merger to the "master" branch or even at every pull request to the selected branch.

1.3.3 Continuous Integration Tools

Continuous integration tools orchestrate most part of the CI/CD pipeline [2]. The CI tool is responsible for taking the source code from the repository (source code management tool), pushing it to the build stage for unit testing and artifact or package generation, execution of test cases, and deployment to the specified environments. A popular example of CI tools are Jenkins, Travis CI, CircleCI etc.

The build stage of the pipeline is declared by developers while deployment stage(s) is handled by the operators. Since the CI tool is a critical component of a DevOps pipeline, it's very important to make sure it's highly available. Examples of fully managed or highly available CI tools are AWS CodePipeline, Travis CI and Gitlab CI (Gitlab Enterprise).

1.3.4 Build Process Tools

Build tools are used to generate deployable packages or artifacts as output using source code as input. The build stage is very developer centred as it varies by programming language. The build tools make continuous delivery possible, by automatically creating a build of pushed developer code and making a deployable version ready.

As earlier mentioned, build stage or tools differ by programming languages. Example of package manager and build tools are as follows: Pip for Python, Maven or Gradle for Java, SBT for Scala, Ruby Gems for Ruby. Maven not only

builds Java code but also produces a .JAR or .WAR artifact as output, this output will then be used for deployment. Some programming languages like Python and Ruby do not need a build stage or build artifacts to be deployed.

Unit testing tools also help developers determine a section or unit of their application meets requirements and behaves as intended. Example of unit testing tools are Mockito and JUNIT for Java, NUnit for .NET and RSpec for Ruby. Selenium help run tests on web applications test cases can be written in Java, python, C#, Ruby, JavaScript and Kotlin.

Using a build tool, declaring build steps and unit testing all help with overall automation of the software development pipeline.

1.3.5 Continuous Deployment Tools

Deployment tools are used to automate software deployments to specified compute destinations. Deployment tools make the CD part of CI/CD possible, either continuous delivery or continuous deployment.

Deployment tools are not limited to carrying out deployment to already provisioned and existing environments but, they are also able to automate infrastructure creation and provisioning on the go. Automated provisioning is synonymous with cloud providers like Amazon Web services, Microsoft Azure and Google cloud. Using IaC tools like AWS CloudFormation or Terraform, infrastructure can be created and provisioned at the deployment stage.

Configuration management tools like Chef, Puppet and Ansible also fall into this category. They help with provisioning or getting the deployment environment ready before the built software is released to these environments.

1.3.6 Continuous Monitoring Tools

Monitoring tools are usually used to track application's non-functional properties, like reliability, resilience, scalability, availability and performance. They are also used to track deployment environment metrics, like CPU utilization, Memory utilization, Free storage etc. Monitoring tools also perform function ranging from alerting, log management to gathering business metrics such as uptime to generate SLA.

Examples of monitoring tools are Splunk, Zabbix, Prometheus, and Nagios. For log management, tools like Graylog and Logstash are used. AWS CloudWatch

also provides monitoring, alarming and log management functionality to solutions deployed in AWS [2].

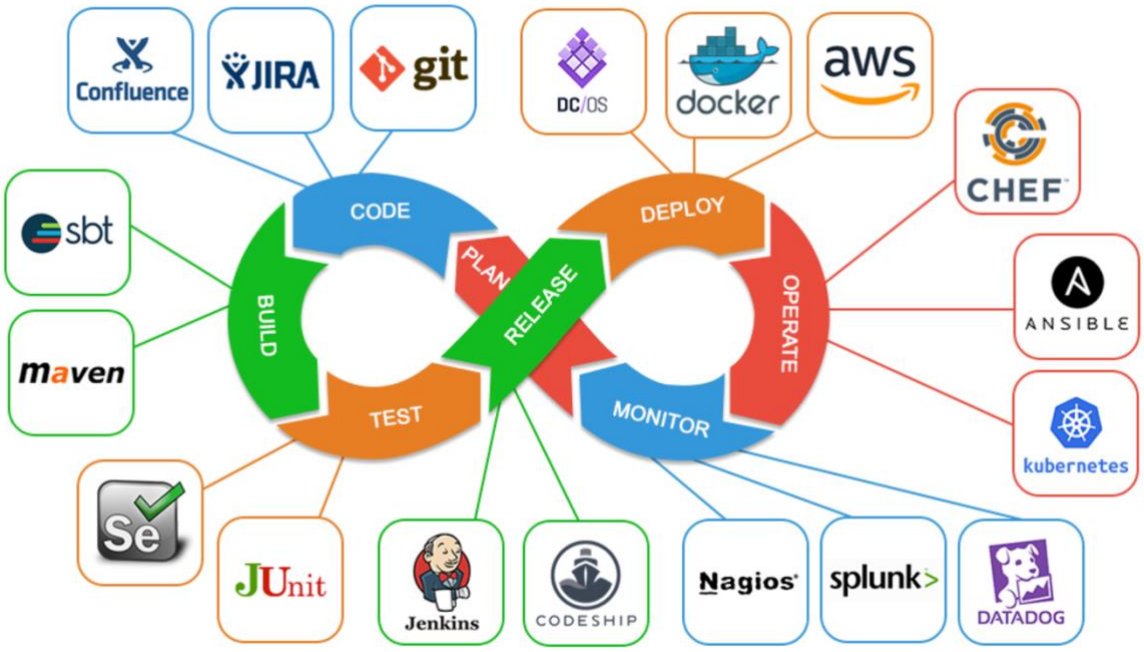

Figure 1.4:DevOps Stages and Example of Tools

1.4 Benefits of DevOps Adoption

The adoption of DevOps in an organization comes with lots of pros and cons. In this section, we'll be analysing the benefits of adopting DevOps.

1. **Speed:**
   Since DevOps is automation heavy, organizations can reduce the time taken to release software or time to market significantly. With a heavy focus on software delivery pipeline automation, organizations will experience reduction in time taken to adapt to market changes and also Innovate at a faster rate. Hence, leading to a competitive advantage over slower organizations.

2. **Rapid Delivery:**
   Continuous deployment and delivery are DevOps practice that automates release of built source code to a specified deployment environment, this leads to an increase in software release as opposed to a traditional method. Frequent software release means organization can increase software

quality faster than ever, bugs can be fixed even more rapidly, and customer expectations can be met as fast as possible.

3. **Reliability:**
   With DevOps practices like continuous testing and continuous delivery which helps with unit testing and different test cases, bugs are discovered and fixed before software gets to the customers. Continuous monitoring also provides real-time insight into software and infrastructure performance. Implementing these components of DevOps helps increase the overall software quality and user trust and experience.

4. **Improved Collaboration and Communication:**
   Benefits from the people side of DevOps practice is also a major plus. Since the practice of DevOps in an organization involves the breaking down of any communication barrier between the development team and the operations team, this leads to an increased in inter-departmental collaboration, there is an increase in trust between personnel. Also, changes can be communicated faster with reduced overhead. Overall increase in collaboration levels, is a plus for the organization.

5. **Scale and Security:**
   Organizations practicing DevOps, are able to operate in a scalable and elastic manner. Using Infrastructure as code to handle deployment or genera infrastructural needs, helps manage deployment environments according to the need of the organization. Automating scaling up and down (auto scaling) also helps handle complex requirements with reduce risk or downtime. Security with majority of DevOps tools can be managed as code (policy as code). This ensures practicing DevOps does not come at the cost of reduce security.

1.5 Challenges of DevOps Adoption

Practicing DevOps comes with some challenges that needs to be taking into consideration, to achieve suitable results.

The first challenge might arise from the lack of conviction from senior management. A lack of proper organizational structure can disrupt proper DevOps adoption, senior managers might be opposed to a new concept especially, when the current mode of operation is perceived to be efficient enough.

The shift to DevOps requires high level of communication and collaboration. For this to work, everyone needs to be onboard to increase the level of collaboration to ensure a successful adoption. Every team needs to embrace this increased collaboration culture from the development team, operations team, product managers, to the marketing team.

DevOps focuses heavily on increased efficiency but, doesn't necessarily reduce cost so much as it grows revenue. Increase in software quality, reliability and speed translates to an increased revenue but the cost of training employees, new tools and cost of increased deployment environments might cause an initial negative net value. All these coupled with the increased responsibility of managing multiple environments are major challenges to DevOps adoption.

1.6 Overview of The Thesis

To compliment previous projects on DevOps, which mostly take the quantitative or the qualitative approach. This project will take a practical approach at DevOps, building actual software development pipelines and practicing DevOps concepts like continuous integration, continuous deployment and continuous delivery.

The Remainder of the thesis is structured as follows:

- In Chapter 2, the tools to be used are discussed. The selected DevOps tools for pipeline (Database pipeline, User interface pipeline, Backend pipeline) are described, reason for selection, pros and cons are enumerated.

- In Chapter 3, the user interface pipeline is explained. Infrastructure as code is explained, deployment process is detailed, software is deployed to selected AWS service.

The database pipeline is explained. Infrastructure as code is explained, deployment process is detailed, software is deployed to selected AWS database service, monitoring is explained.

The backend pipeline is also discussed. Infrastructure as code is explained, deployment process is detailed, software is deployed to selected AWS container service.

- In Chapter 4, the conclusions made from the thesis are explained. Possible improvements, and the possibility for future research areas are also discussed.

2 Start of the Art

2.1 DevOps Tools Used

The practical section of this project shows how DevOps concepts like continuous integration, continuous delivery and deployment can be implemented in different parts of a distributed software. This project specifically focuses on using DevOps cloud tools from Amazon web service.

2.1.1 GitHub (Version Control)

GitHub is a Git hosting repository that enables developers to collaborate better. With features like pull requests, code review, issues (ticketing system), developers can seamlessly work on the same code repository without conflicts.

Developers can work on the different branches of the same repository, commit code changes to save them, create a pull request to the main branch, code changes can be discussed, or peer reviewed before the pull request is merged.

A repository, also referred to as a repo. Repos are Git projects containing files and folders related to a development project, along with each file's revision history. In Repositories, new branches can be created, pull requests and mergers will also occur within a repository. The changelog shows the change history of files within the repository.

GitHub is used to store and version all code changes for this project.

2.1.2 AWS CloudFormation

CloudFormation is an infrastructure as code tool that helps with the creation AWS and third-party resources, provision them rapidly and consistently, and manage them throughout their creation, updates and deletion.

Resources are declared in a template in either yaml or json. This allows for stacks to be replicated in a controlled and predictable way. CloudFormation can used to be manage simple or very complex stacks.

All infrastructural resources in this project are declared using a CloudFormation template in yaml.

2.1.3 AWS CodePipeline

CodePipeline is a continuous integration delivery tool, that helps automate software release pipelines, providing fast and reliable infrastructure and application. It gets triggered by a source control, and handles the orchestration of build, test and deploy phases of the release process.

It's a fully managed AWS service, which integrates seamlessly with GitHub or any other third-party service.

CodePipeline is used to automate and orchestrate all source code build, test and deployments phases for this project.

2.1.4 AWS CodeBuild

AWS CodeBuild is a continuous integration tool that compiles source code, runs tests, and provides deployable software packages called artifacts. CodeBuild is fully managed by AWS, there is no need to create, manage or scale build servers, it's all taken care of automatically.

CodeBuild can handle multiple builds simultaneously, different CI/CD pipelines can run at the build stage in parallel. Builds can either be carried out in preconfigured environments, AWS CodeBuild provides build environments for Java, Python, Node.js, Ruby, Go, Android, .NET Core for Linux, and Docker. Builds can also be done in user customized environments to provide a bespoke solution.

All software builds made in the project was built using AWS CodeBuild.

2.1.5 AWS CodeDeploy

CodeDeploy is another DevOps tool, fully managed by AWS. It automates software deployment to different AWS compute services like servers, containers, serverless and on premises environments. Using a deployment service like code deploy, prevents downtime during software releases, improves release speeds while handling extremely complex scenarios.

AWS CodeDeploy is platform and language agnostic and works with any application. CodeDeploy automates deployments to the containerized environment used in this project.

2.2 Other Tools

This project shows how different types of software delivery pipelines can be automated. Static website deploys to AWS S3 and AWS CloudFront, database pipeline to Amazon RDS and AWS ApiGateway, Backend application pipeline to AWS ECS.

2.2.1 Amazon RDS

Amazon Relational Database Service makes the process of setting up, operating, and scaling a relational database in the cloud easy. It provides a managed database service, i.e., time consuming tasks like hardware provisioning, database setup, patching and backups while also being cost efficient. More focused can be placed on delivering the best user experience to application users.

RDS supports six popular database engines including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server. It can also be customized to suit the exact use case, selections range from memory optimized instance, performance optimized to I/O optimized instances.

2.2.2 Amazon API Gateway

APIs act as an entry point for applications to access data or business logic. Amazon API Gateway eases the process of creating, publishing, maintaining, monitoring and securing APIs at any scale. It's a fully managed service that supports creation of RESTful APIs and WebSocket APIs that enable real-time two-way communication applications.

API Gateway is environment agnostic, it supports containerized and serverless workloads, as well as web applications. It handles all the necessary tasks involved in accepting and processing multiple concurrent API calls, CORS support, traffic management, authorization and access control, throttling, API version management and monitoring.

For this project, API Gateway will be used to GET data stored in the database instead of directly reading from the database endpoint. This not only provides a secure access to the database instance protected with an API key, it also allows the database instance to remain in a private subnet without being exposed to attacks over the internet.

2.2.3 Amazon Simple Storage Service (Amazon S3)

Amazon S3 is an object storage service that offers data durability, scalability, availability, security and performance. S3 offers a plethora of use cases like, storing static website components, mobile application, for backup and restore, enterprise applications, big data analytics and archive. S3 is a very durable object storage, designed for 99.9999999999% (11 9's) of durability to be specific.

S3 website feature is implemented as delivery point for the "static website pipeline". S3 bucket will be configured to serve website files upon every request. The solution is one of the easiest and cheapest ways to deploy a static website as there are no servers or containers involved.

2.2.4 Amazon CloudFront

Amazon CloudFront is AWS's own content delivery network (CDN) service the securely delivers global content like, videos, data, applications and APIs to users with high transfer speeds and very low latency.

CloudFront also offers advance security capabilities which includes field level encryption and HTTPS support. Integration with AWS services like AWS Shield, AWS Web Application Firewall for high level protection is seamless.

The "static website pipeline" serves website files using CloudFront, to make up for the inability of S3 bucket website to perform compute required optimization.

2.2.5 Amazon Elastic Container Service (Amazon ECS)

Containers are a standard unit of software that packages up source code and all its dependencies, so the application runs reliably and quickly independent of computing environment. Containers are lightweight and a portable variant to full fledge servers.

Amazon ECS is a fully managed container orchestrator. It's a fully managed service that provides security, reliability and scalability to mission critical and sensitive applications.

This project uses Amazon ECS as a development environment for the "Dynamic website pipeline".

3 CI/CD Pipelines

3.1 Database Pipeline

3.1.1 Architecture (Infrastructure as Code)

This project implements a security cautious database architecture. The Postgres database is deployed in a private subnet and only accessed through API Gateway. API Gateway resources are secured with an API key. For the purpose of this infrastructure, the consumer of the API is a frontend application. The application performs only a ready action on the database.

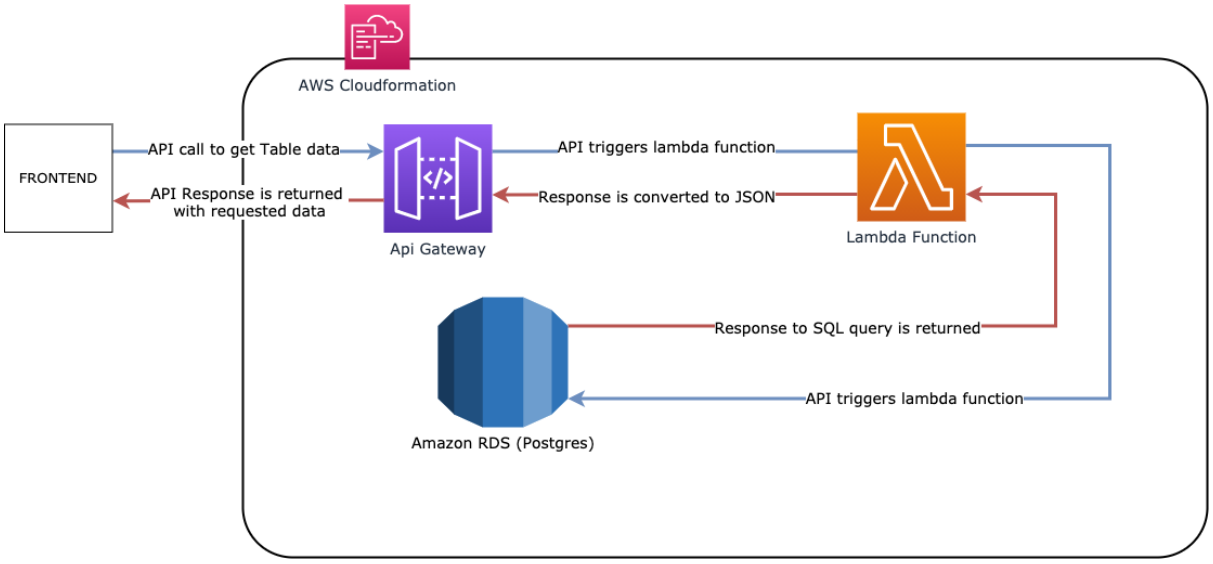

Figure 3.1:Database stack architecture

The sequence starts when the application makes an API call through Amazon API Gateway, providing an API key. After validating the API key, API Gateway triggers a Lambda function.

The Lambda function executes a python script to run a SQL query, for example a SELECT * from the appropriate table. The response from the database is returned as a key-value pair

The Lambda function takes this response from the database and delivers it to API Gateway as a JSON payload. This payload is then delivered to the application.

The whole infrastructure is declared as a CloudFormation template. This allows for versioning and easy replication of our infrastructure. A practice applauded with the practice of DevOps.

3.1.2 CI/CD Pipeline

The delivery pipeline is customised to automate the creation of schemas, tables and send user feedback after completion.

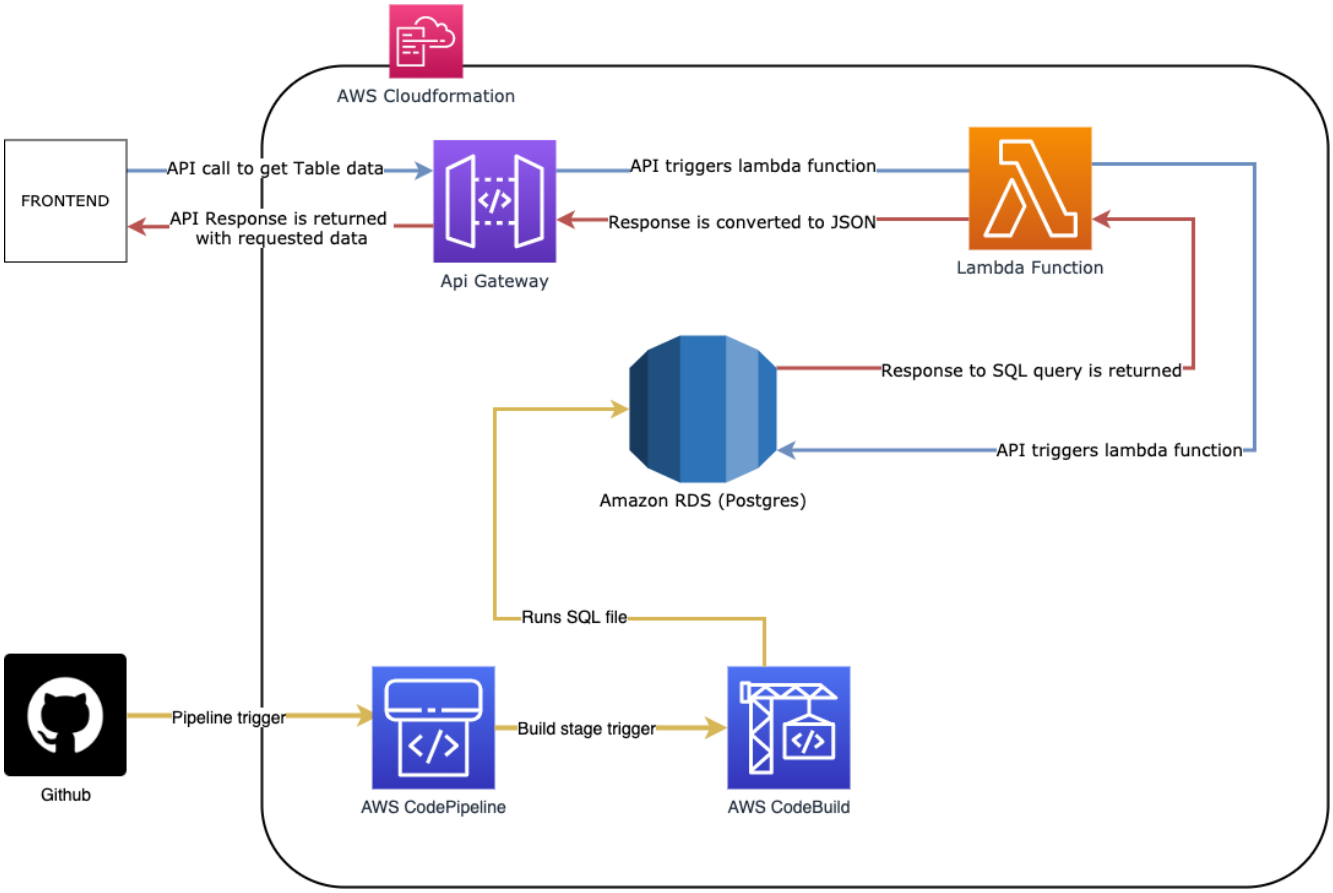

Figure 3.2:Database pipeline

The CI/CD pipeline is trigger by developer action to the GitHub branch specified in the CloudFormation template, CodePipeline triggers the build engine in a CodeBuild instance. Since this CI/CD is really not a traditional software or application that needs to be compiled or packaged, this stage simply automates SQL queries that would have otherwise had to be run manually.

The build stage:

- Installs a Postgres client
- Connects to the RDS (Postgres) instance created, taking required parameters like database host, database username and database password from the environmental variable declared in the CloudFormation template
- Runs the SQL queries declared in the specified files
- Runs a SELECT * query on created tables to give user feedback

A successful build means we've successfully automated the process of running SQL commands against a database instance.

3.1.3 Deployment

The CloudFormation template was deployed through the AWS console with appropriate parameters.

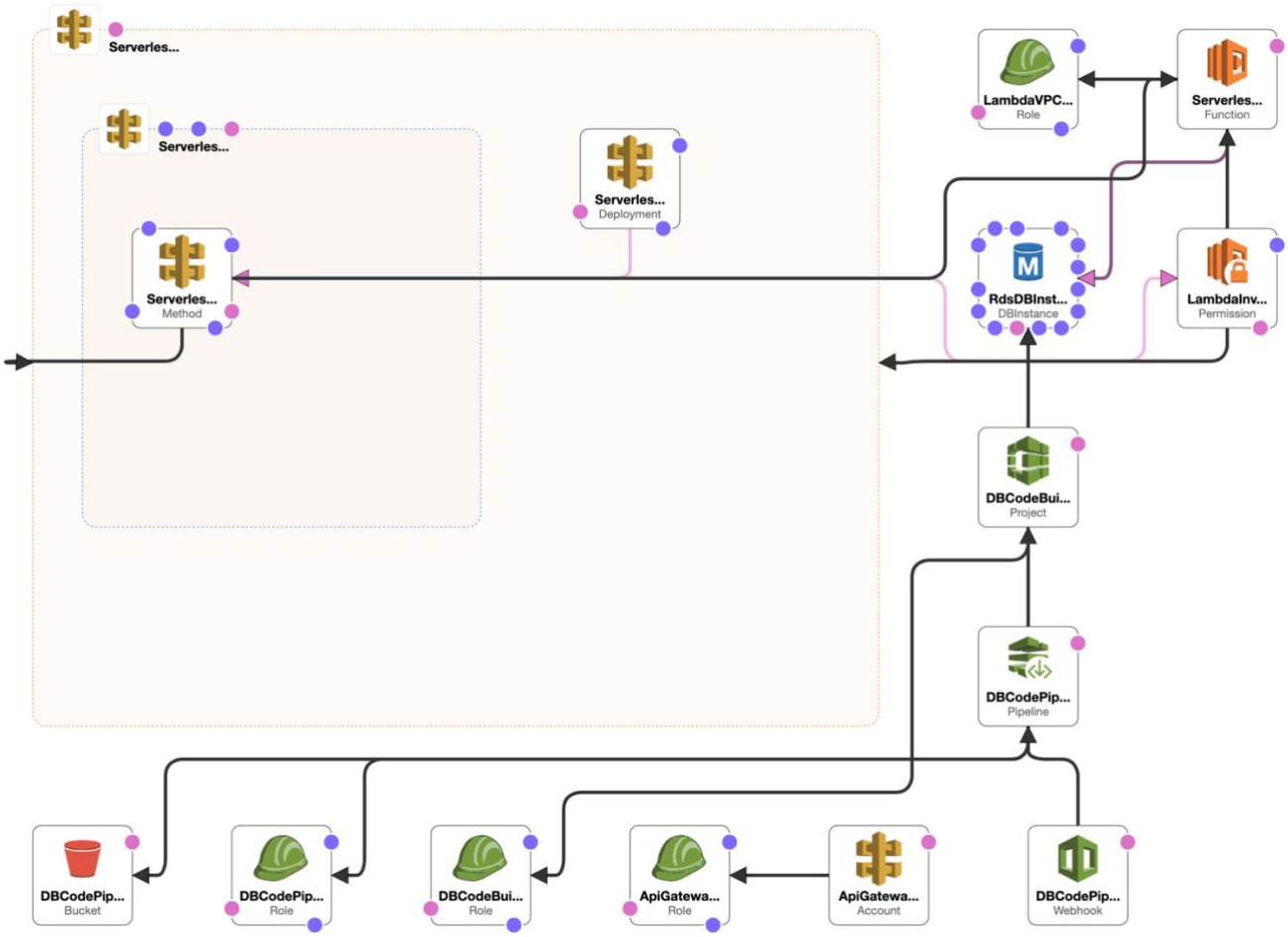

Figure 3.3:Database CloudFormation deployment

Immediately after stack creation was successful, the CI/CD pipeline was triggered by a merger to the master branch in the GitHub Repository.

```
228
229 [Container] 2021/01/22 21:48:42 Running command export PGPASSWORD=$Password
230
231 [Container] 2021/01/22 21:48:42 Running command psql --host=$DBEndPoint --dbname=$DBname --port=$Port --username=$DBUser  < "postgresql-commands/moxa-
    devices.sql"
232 NOTICE:  schema "test" already exists, skipping
233 CREATE SCHEMA
234 NOTICE:  relation "customers" already exists, skipping
235 CREATE TABLE
236 INSERT 0 5
237 WARNING:  there is no transaction in progress
238 COMMIT
239  customerid |       customername       |    contactname     |           address            |    city     | postalcode | country
240 ------------+--------------------------+--------------------+------------------------------+-------------+------------+---------
241           1 | Alfreds Futterkiste      | Maria Anders       | Obere Str. 57                | Berlin      | 12209      | Germany
242           2 | Ana Trujillo Emparedados | Ana Trujillo       | Avda. de la Constitución 2222 | México D.F  | 05021      | Mexico
243           3 | Antonio Moreno Taquería  | Antonio Moreno     | Mataderos 2312               | México D.F. | 05023      | Mexico
244           4 | Around the Horn          | Thomas Hardy       | 120 Hanover Sq.              | London      | WA1 1DP    | UK
245           5 | Berglunds snabbköp       | Christina Berglund | Berguvsvägen 8               | Luleå       | S-958 22   | Sweden
246 (5 rows)
247
248
249 [Container] 2021/01/22 21:48:42 Running command echo Deploy started on `date`
250 Deploy started on Fri Jan 22 21:48:42 UTC 2021
251
252 [Container] 2021/01/22 21:48:42 Running command echo Deploying to $ENV_NAME environment...
253 Deploying to dev environment...
254
255 [Container] 2021/01/22 21:48:42 Phase complete: BUILD State: SUCCEEDED
256 [Container] 2021/01/22 21:48:42 Phase context status code:  Message:
257 [Container] 2021/01/22 21:48:42 Entering phase POST_BUILD
258 [Container] 2021/01/22 21:48:42 Phase complete: POST_BUILD State: SUCCEEDED
259 [Container] 2021/01/22 21:48:42 Phase context status code:  Message:
260 [Container] 2021/01/22 21:48:42 Expanding base directory path: .
261 [Container] 2021/01/22 21:48:42 Assembling file list
262 [Container] 2021/01/22 21:48:42 Expanding .
263 [Container] 2021/01/22 21:48:42 Expanding file paths for base directory .
264 [Container] 2021/01/22 21:48:42 Assembling file list
265 [Container] 2021/01/22 21:48:42 Expanding **/*
266 [Container] 2021/01/22 21:48:42 Found 11 file(s)
267 [Container] 2021/01/22 21:48:42 Phase complete: UPLOAD_ARTIFACTS State: SUCCEEDED
268 [Container] 2021/01/22 21:48:42 Phase context status code:  Message:
269
```

Figure 3.4:Database build stage

The build stages executes the SQL queries in the files on the specified path. It returns the content on the table after a successful execution.

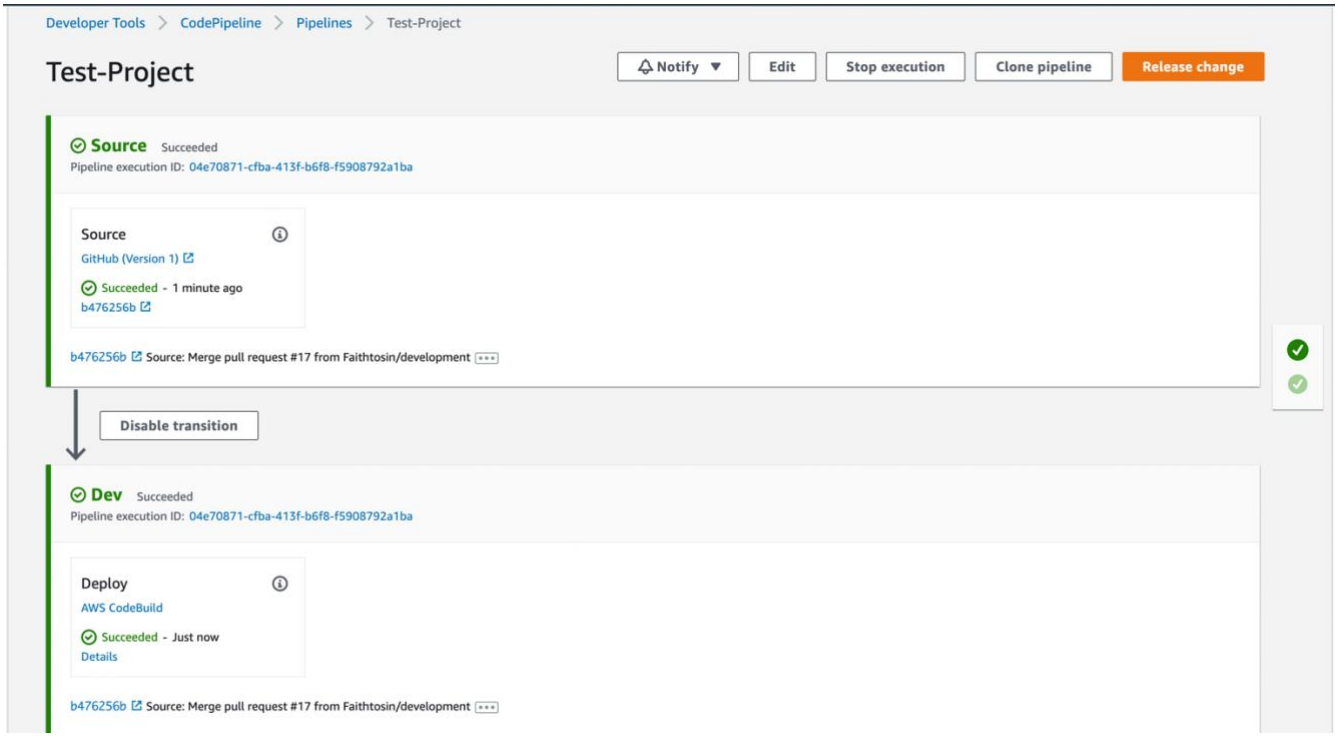

Figure 3.5:Database pipeline stages

The AWS console provides a visual feedback detailing the status of each stage of the pipeline.

3.2 Static Website Pipeline

3.2.1 Architecture (Infrastructure as Code)

The Static website is deployed as an S3 website, this allows static website files to be deployed in a very cheap manner while enjoying benefits of Amazon S3 like versioning, policy-based security and high durability. The S3 bucket uses a lifecycle policy to delete old and not needed files to keep the bucket clean and ordered while keeping storage cost low.

The website is built with latency in mind. The architecture makes use of a CDN which allows the end users to get response to request significantly faster than normal. The CDN used is Amazon CloudFront, it makes use of AWS edge locations to cache most requested files and deliver them to users in fast secure manner.

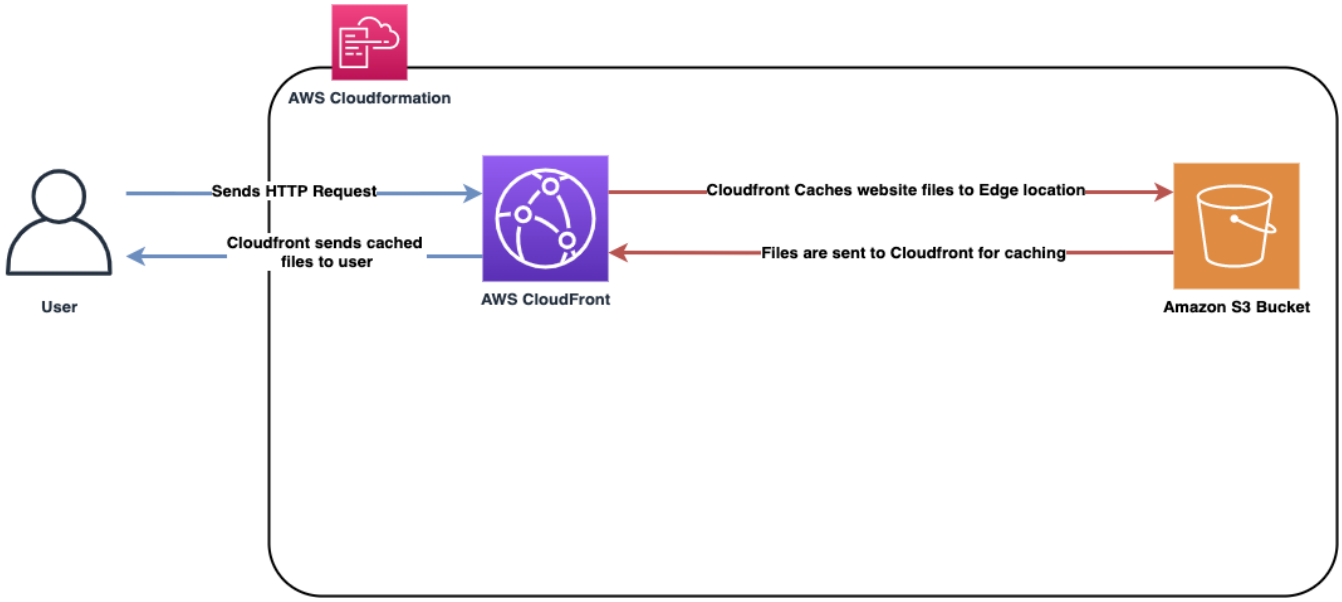

Figure 3.6:Static website stack

CloudFront is set to update its version of website files every 1 hour, the process is called invalidation. After every deployment, cached files will be invalidated as part of the build process and new ones copied over at the first request.

3.2.2 CI/CD Pipeline

The delivery pipeline is built to automate the website deployment process.

The CI/CD pipeline is triggered by developer action to the GitHub branch specified in the CloudFormation template, CodePipeline triggers the build engine in a CodeBuild instance.

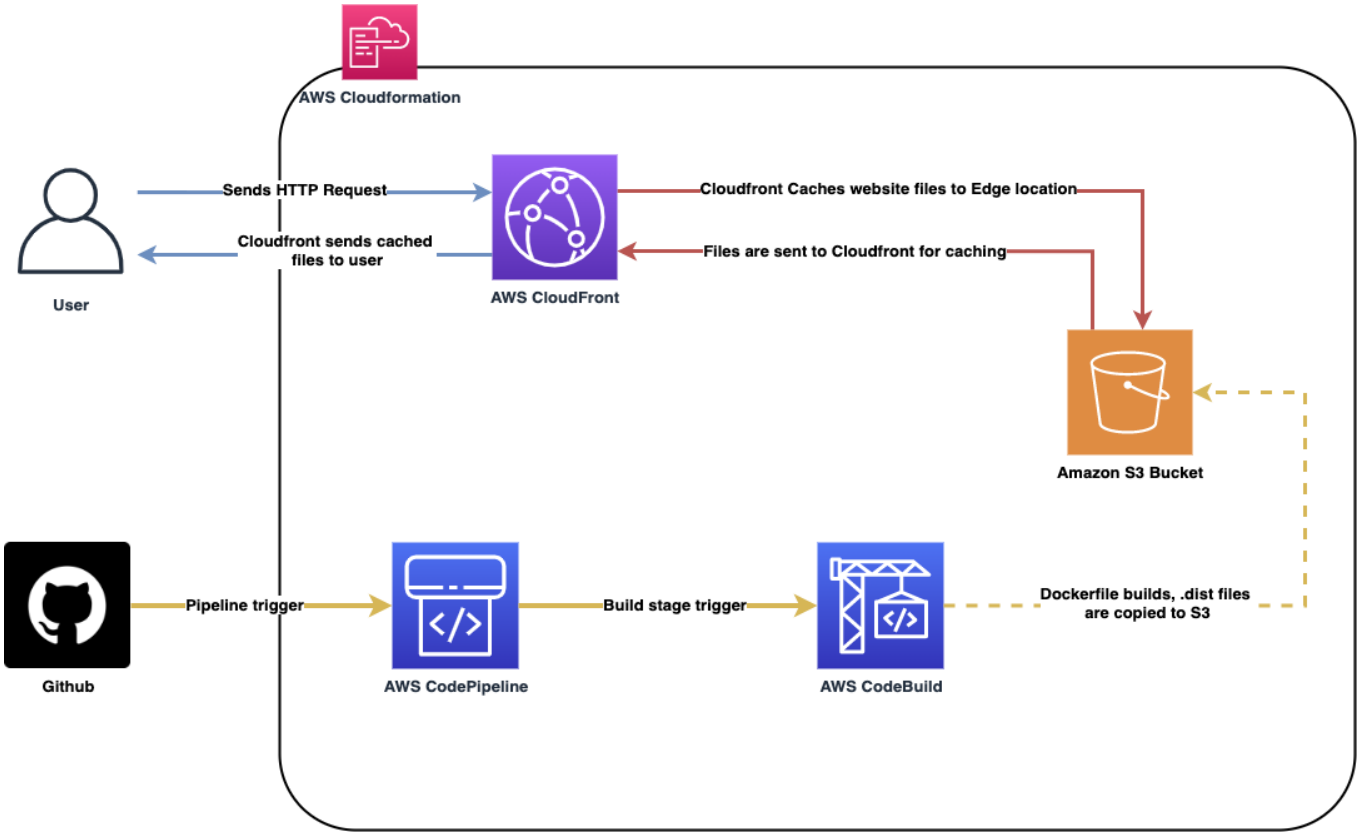

Figure 3.7:Static website pipeline

CodeBuild is set to build using the Dockerfile declared in the repository. After the build, the build files located in the. dist folder in the case (angular application) is copied from the build container to the website S3 bucket.

After the build stage, the CloudFront files are invalidated on the last build step using an AWS cli command.

3.2.3 Deployment

The CloudFormation template was deployed through the AWS console with appropriate parameters.

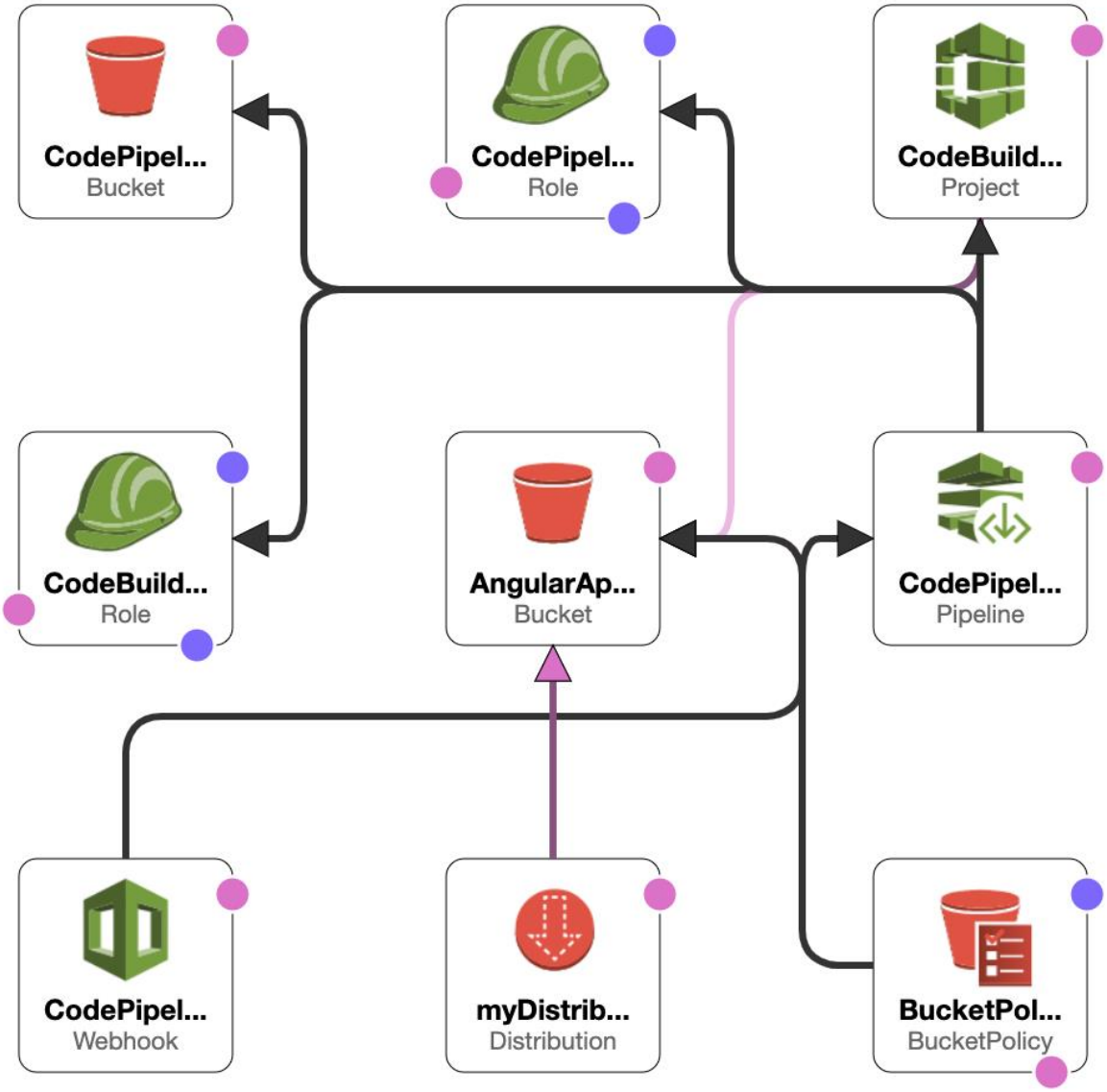

Figure 3.8:Static website CloudFormation resources

Immediately after stack creation was successful, the CI/CD pipeline was triggered by a merger to the master branch in the GitHub Repository.

```
973 ·[1AHeadlessChrome 0.0.0 (Linux 0.0.0): Executed 5 of 5 SUCCESS (0 secs / 0.145 secs)
974 ·[1AHeadlessChrome 0.0.0 (Linux 0.0.0): Executed 5 of 5 SUCCESS (0.151 secs / 0.145 secs)
975 TOTAL: 5 SUCCESS
976 TOTAL: 5 SUCCESS
977
978 [Container] 2021/01/30 20:06:56 Phase complete: BUILD State: SUCCEEDED
979 [Container] 2021/01/30 20:06:56 Phase context status code:  Message:
980 [Container] 2021/01/30 20:06:56 Entering phase POST_BUILD
981 [Container] 2021/01/30 20:06:56 Running command aws s3 sync ./dist/angular-devops s3://ui-bucket-develop --acl 'public-
    read' --delete
982 Completed 2.1 KiB/320.4 KiB (50.9 KiB/s) with 7 file(s) remaining
983 upload: dist/angular-devops/3rdpartylicenses.txt to s3://ui-bucket-develop/3rdpartylicenses.txt
984 Completed 2.1 KiB/320.4 KiB (50.9 KiB/s) with 6 file(s) remaining
985 Completed 2.7 KiB/320.4 KiB (26.4 KiB/s) with 6 file(s) remaining
986 upload: dist/angular-devops/index.html to s3://ui-bucket-develop/index.html
987 Completed 2.7 KiB/320.4 KiB (26.4 KiB/s) with 5 file(s) remaining
988 Completed 255.4 KiB/320.4 KiB (2.4 MiB/s) with 5 file(s) remaining
989 upload: dist/angular-devops/main.602e8da16c739e437756.js to s3://ui-bucket-develop/main.602e8da16c739e437756.js
990 Completed 255.4 KiB/320.4 KiB (2.4 MiB/s) with 4 file(s) remaining
991 Completed 313.6 KiB/320.4 KiB (2.9 MiB/s) with 4 file(s) remaining
992 upload: dist/angular-devops/polyfills.38cfeb63b91a963d9fbf.js to s3://ui-bucket-develop/polyfills.38cfeb63b91a963d9fbf.js
993 Completed 313.6 KiB/320.4 KiB (2.9 MiB/s) with 3 file(s) remaining
994 Completed 313.7 KiB/320.4 KiB (2.9 MiB/s) with 3 file(s) remaining
995 upload: dist/angular-devops/styles.105649057273743297b8.css to s3://ui-bucket-develop/styles.105649057273743297b8.css
996 Completed 313.7 KiB/320.4 KiB (2.9 MiB/s) with 2 file(s) remaining
997 Completed 319.0 KiB/320.4 KiB (2.8 MiB/s) with 2 file(s) remaining
998 upload: dist/angular-devops/favicon.ico to s3://ui-bucket-develop/favicon.ico
999 Completed 319.0 KiB/320.4 KiB (2.8 MiB/s) with 1 file(s) remaining
1000 Completed 320.4 KiB/320.4 KiB (1.3 MiB/s) with 1 file(s) remaining
1001 upload: dist/angular-devops/runtime.ec2944dd8b20ec099bf3.js to s3://ui-bucket-develop/runtime.ec2944dd8b20ec099bf3.js
1002
1003 [Container] 2021/01/30 20:07:01 Phase complete: POST_BUILD State: SUCCEEDED
1004 [Container] 2021/01/30 20:07:01 Phase context status code:  Message:
1005 [Container] 2021/01/30 20:07:02 Expanding base directory path: .
1006 [Container] 2021/01/30 20:07:02 Assembling file list
1007 [Container] 2021/01/30 20:07:02 Expanding .
1008 [Container] 2021/01/30 20:07:02 Expanding file paths for base directory .
1009 [Container] 2021/01/30 20:07:02 Assembling file list
1010 [Container] 2021/01/30 20:07:02 Expanding **/*
1011 [Container] 2021/01/30 20:07:02 Found 27554 file(s)
1012 [Container] 2021/01/30 20:07:10 Phase complete: UPLOAD_ARTIFACTS State: SUCCEEDED
1013 [Container] 2021/01/30 20:07:10 Phase context status code:  Message:
1014
```

Figure 3.9:Static website Build stage

The build stage packages the source code into deployable artifacts. These artifacts are copied to the destination S3 bucket.

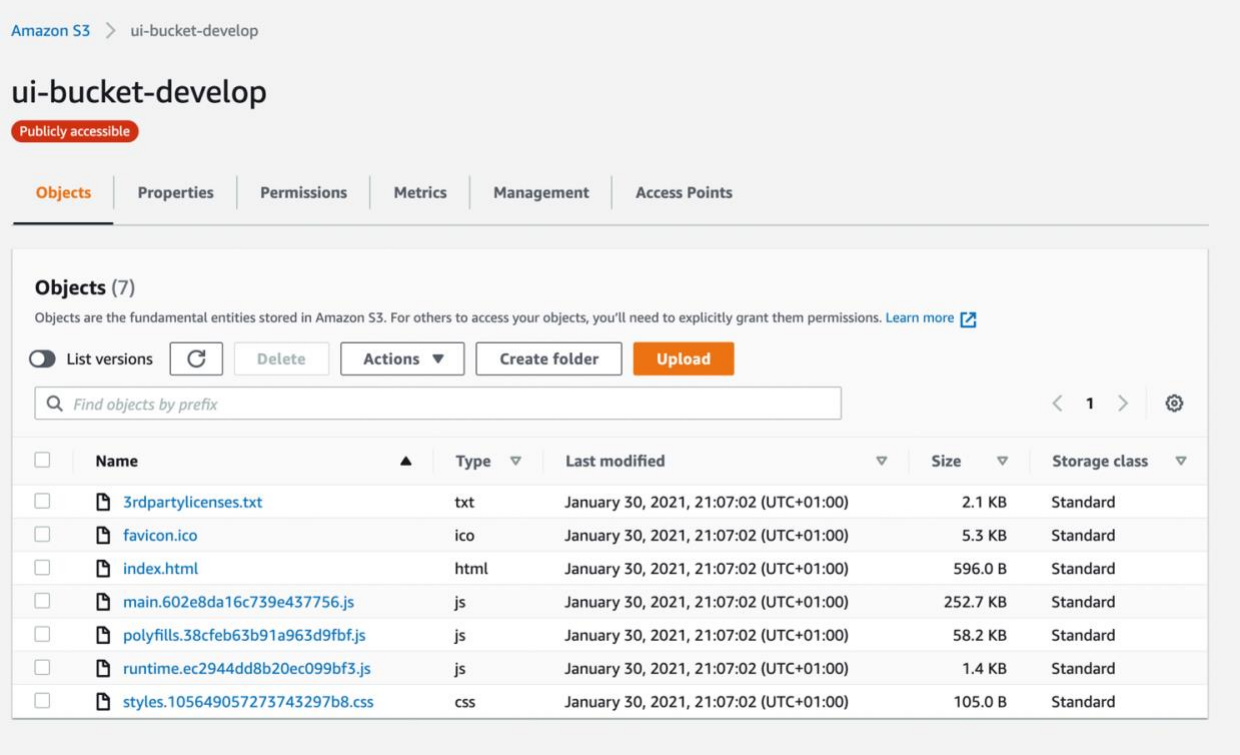

Figure 3.10:S3 bucket with website files

With the built source code copied to the specified destination, all stages of the CI/CD pipeline are declared successful.

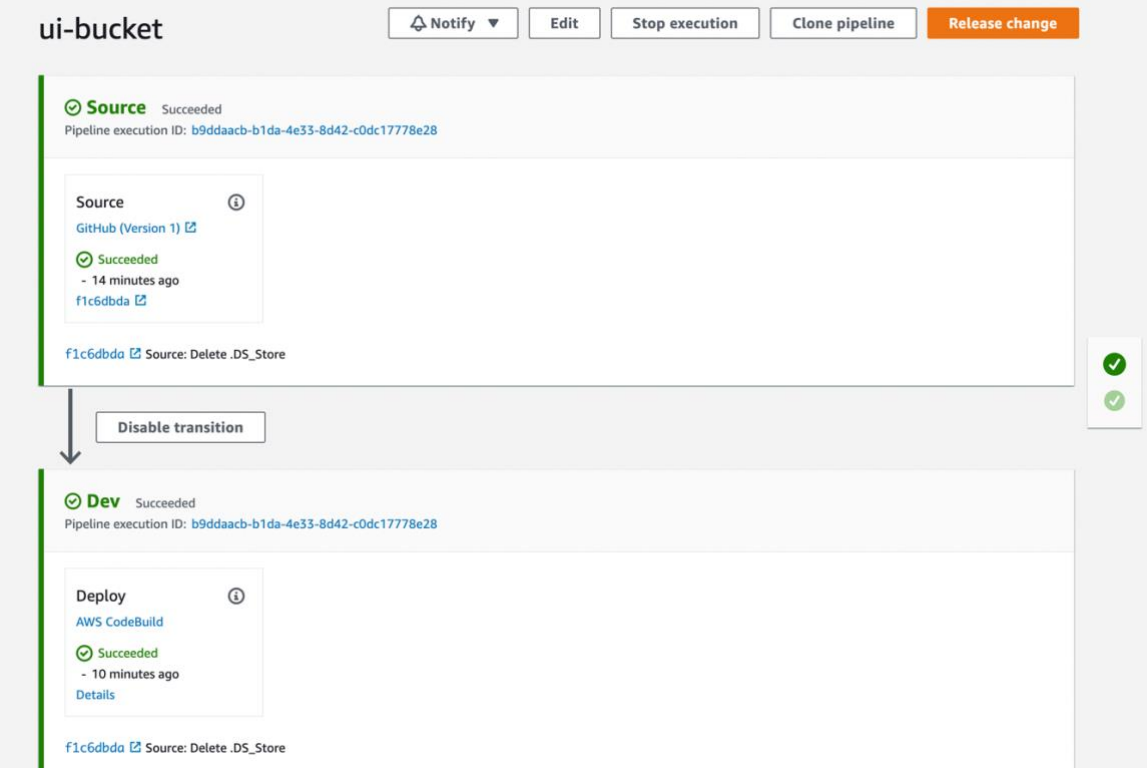

Figure 3.11:Static website CI/CD pipeline

The angular application is accessed using the CloudFront endpoint.

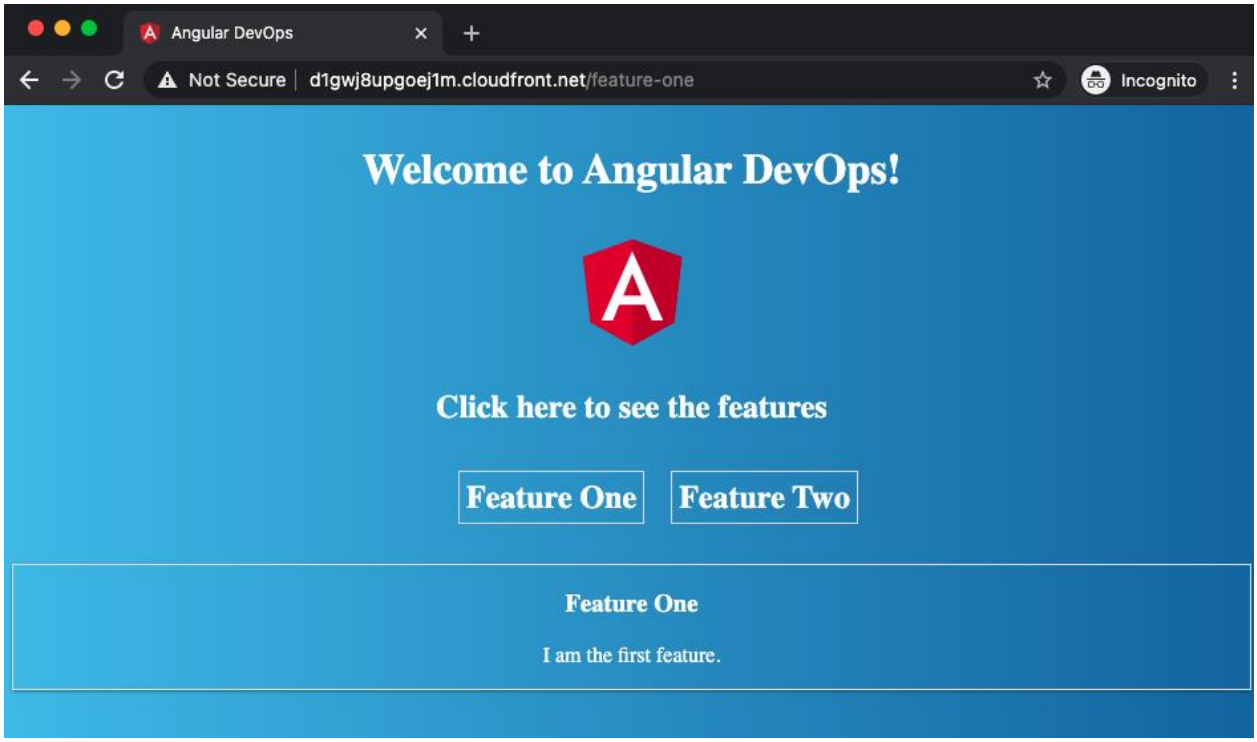

Figure 3.12:Angular web application

3.3 Dynamic website pipeline

3.3.1 Architecture (Infrastructure as Code)

The dynamic web application is deployed in a containerized environment. The request from users is passed through CloudFront, this reduces latency to a bare minimum by caching frequently requested file at edge locations closest to the user. CloudFront forwards this request to an application load balancer. The load balancer does exactly what it says, it distributes load according to declared set of rules to a specified target. The target in this case are containers in a target group. By distributing load evenly among these containers, resources are maximized without stressing any particular container. Containers also connect to a database, to store and retrieve needed data.

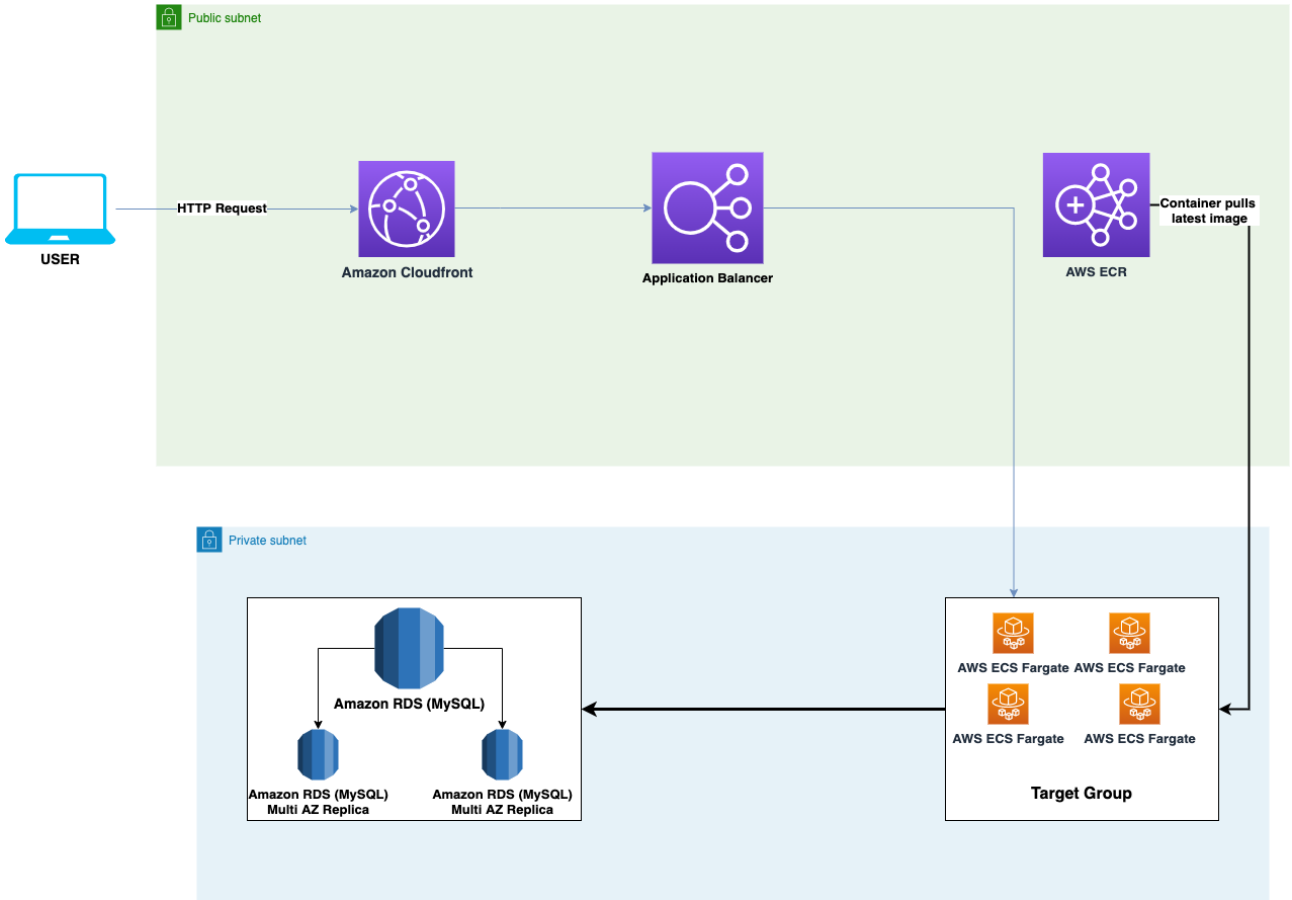

Figure 3.13:Dynamic website stack

Amazon ECR is a container registry housing the docker images needed by the containers to provision packaged code.

3.3.2 CI/CD Pipeline

The delivery pipeline for the dynamic website uses 3 tools in this case. CodeDeploy is needed to automate deployment to a compute service, in this case Amazon ECS.

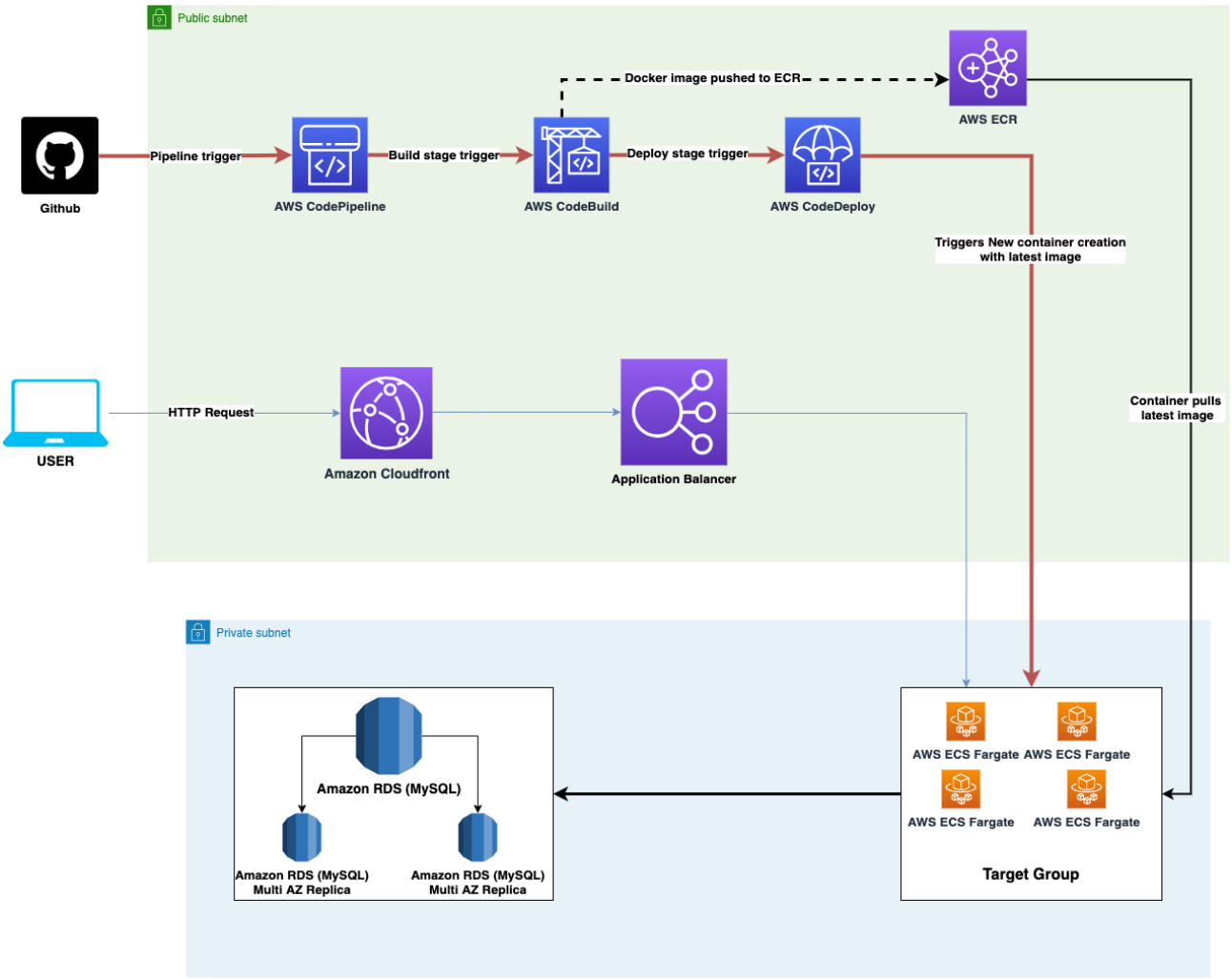

Figure 3.14:Dynamic website pipeline

The CI/CD pipeline is triggered by developer action to the GitHub branch specified in the CloudFormation template, CodePipeline triggers the build engine in a CodeBuild instance. CodeBuild builds the docker image, pushes it to ECR with the tag “latest”.

After a successful build, CodeBuild then triggers the next stage “Deployment stage”. Using artifacts from the build stage, CodeDeploy uses this information

to deploy new containers. The containers are spun up, health checks are performed. if passed, traffic is directed towards the newly deployed containers.

3.3.3 Deployment

As with other delivery pipelines in this project, CloudFormation template was deployed through the AWS console with appropriate parameters.

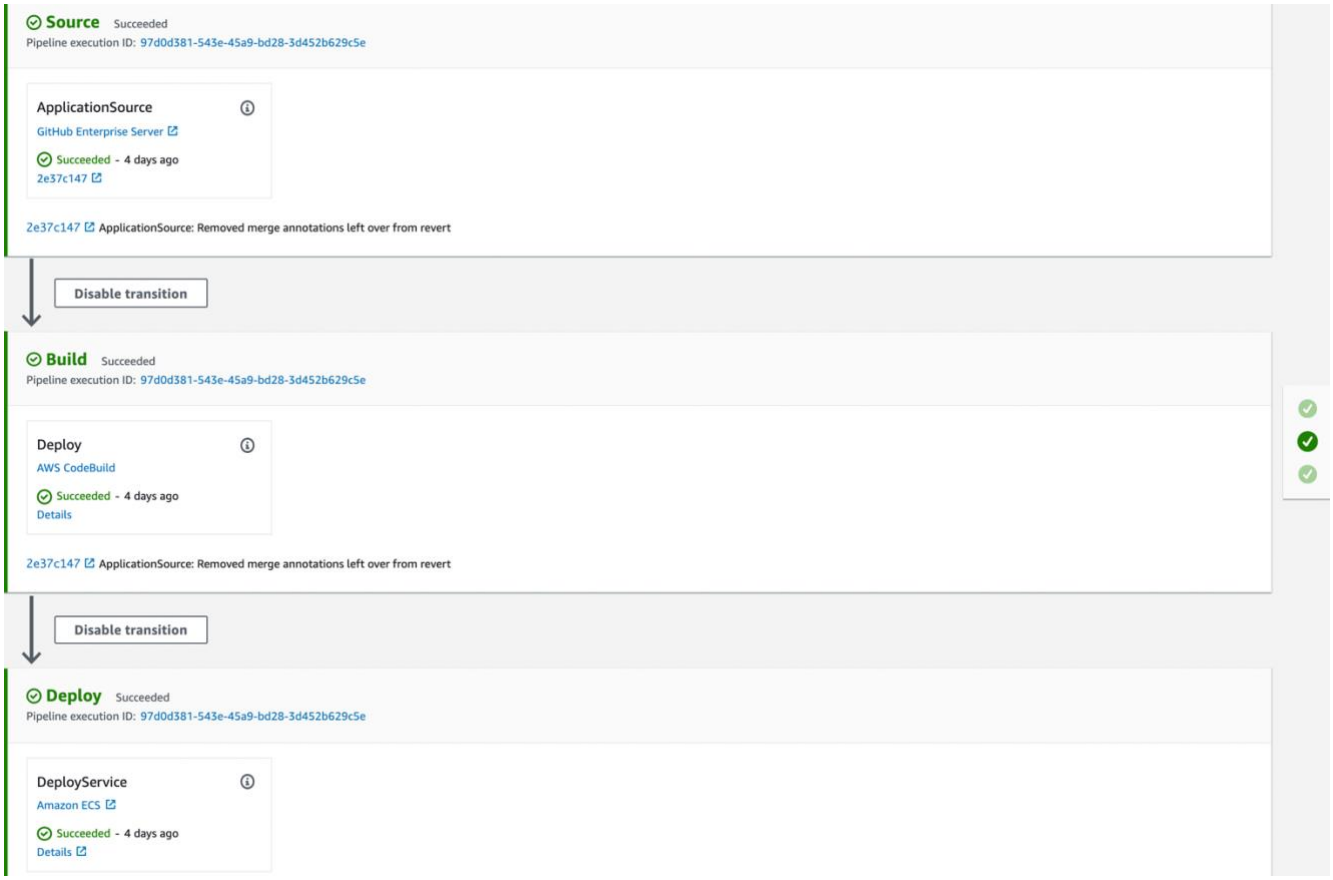

Figure 3.15:Dynamic website CI/CD pipeline

The visual feedback shows, the 3 stages of the pipeline ran successfully all other 5mins.

4 Conclusion

4.1 Results achieved

In summary, the technical section of this project tests the adoption of CI/CD pipelines with different use cases. Results achieved from implementing the DevOps process was compared to using a traditional software delivery process:

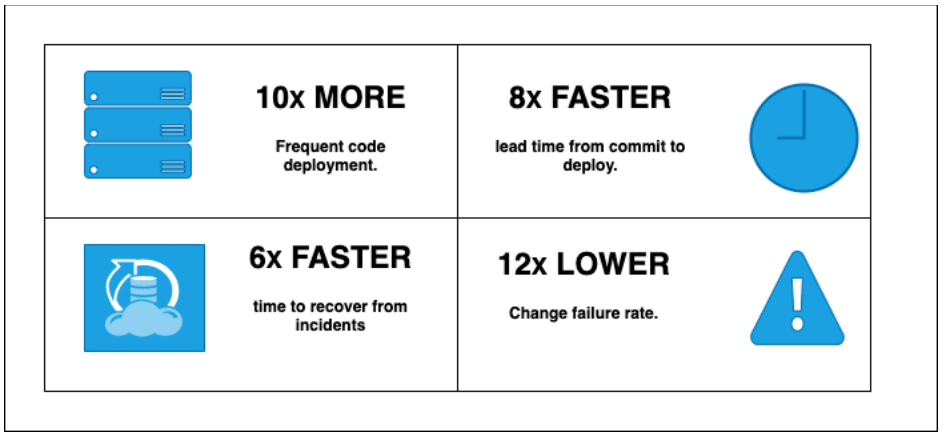

Figure 4.1:Results achieved

- Change failure rate (Lower is better): this is the likelihood for code changes to fail during the deployment process. Using a continuous delivery process, failure rate drastically reduced as opposed to a traditional process.

- Incident Recovery Time (Lower is better): This is the time taken to recover from general incidents, this includes incident experienced during new deployment in production. With the current modernized architecture, features like roll back and health checking aids speedy recovery, reducing down time. It's also important to make sure delivery is not being sped up at the expense of negative customer experience.

- Time from commit to deploy (lower is better): This is the total time taken to deliver software from source control management to complete deployment. Total time taken for commit to become and actual product reduced drastically with the use of a CI/CD pipeline.

- Number of code deployments (higher is better): This is measure of how often a software deploys to production. It is a batch size indicator; smaller batch sizes show greater market agility. Could also be translated to number of features released or bug fixes pushed. Practicing DevOps technically means an increase to this metric.

4.2 Future Improvements

While this project experimented with CI/CD pipelines in different scenarios, there still lots of improvements to be done on the pipelines built in this project, and in DevOps research generally.

Involving a monitoring tool provides possibility to track the application's non-functional properties, like reliability, resilience, scalability, availability and performance. They are also used to track deployment environment metrics, like CPU utilization, Memory utilization, Free storage etc. Adding a monitoring tool gives visibility into the stack.

Cost optimization is also a big factor, deploying solutions to the cloud can be costly if not properly optimized. Cloud financial management (FinOps) helps identify the most efficient way for teams to manage their cloud costs, where everyone takes ownership of their cloud usage supported by a central best-practices group. Cross-functional teams work together to enable faster delivery, while at the same time gaining more financial and operational control.

The inclusion of Automated Testing to the pipeline drastically reduces bugs and increase quality of shipped software. Inclusion of Unit testing makes sure the code functions as intended. Integrations test can help test interconnectivity of infrastructural components. GUI testing can also help test the deployed application's functionality.

References

[1] “What is DevOps? - Amazon Web Services (AWS),” *Amazon Web Services, Inc.* https://aws.amazon.com/devops/what-is-devops/ (accessed Dec. 26, 2020).

[2] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A Survey of DevOps Concepts and Challenges,” *ACM Comput. Surv.*, vol. 52, no. 6, p. 127:1-127:35, Nov. 2019, doi: 10.1145/3359981.