Contents

Overview

This project is an introduction on how to use Amazon Web Services (AWS) in DevOps. AWS resources can decrease time to market and reduce costs for companies. The paper will discuss a specific tool called AWS code pipeline, it is a relatively new tool which was released in July 2015.

We used AWS CodePipeline to automat your software release process, allowing us to rapidly release new features to users. With CodePipeline, we can quickly iterate on feedback and get new features to the users faster.

Automating the build, test, and release process allowed us to test each code change and catch bugs while they are small and simple to fix quickly and easily. We can ensure the quality of our application or infrastructure code by running each change through our staging and release process.

In today's world applications must evolve quickly for customers. Improving and releasing software at a fast pace to customer needs to be at the core of every business. Making time and agility to market essential to maintaining your competitive advantage. Companies that can rapidly deliver updates to applications and services, innovate and change faster to adapting changing markets which give better results to business and customers.

With AWS code pipeline, companies can deliver value to their customers quickly and safely.

Continuous Integration (CI)

Continuous integration (CI) is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. CI most often refers to the build or integration stage of the software release process and requires both an automation component (for example a CI or build service) and a cultural component (for example learning to integrate frequently). The key goals of CI are to find and address bugs more quickly, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous integration focuses on smaller commits and smaller code changes to integrate. A developer commits code at regular intervals, at minimum once a day. The developer pulls code from the code repository to ensure the code on the local host is merged before pushing to the build server. At this stage the build server runs the various tests and either accepts or rejects the code commit.

The basic challenges of implementing CI include more frequent commits to the common codebase, maintaining a single source code repository, automating builds, and automating testing. Additional challenges include testing in similar environments to production, providing visibility of the process to the team, and allowing developers to easily obtain any version of the application.

Continuous Delivery (CD)

Continuous delivery (CD) is a software development practice where code changes are automatically built, tested, and prepared for production release. It expands on continuous integration by deploying all code changes to a testing environment, a production environment, or both after the build stage has been completed. Continuous delivery can be fully automated with a workflow process or partially automated with manual steps at critical points. When continuous delivery is properly implemented, developers always have a deployment-ready build artifact that has passed through a standardized test process. With continuous deployment, revisions are deployed to a production environment.

CD can help you discover and address bugs early in the delivery process before they grow into larger problems later. Our team can easily perform additional types of code tests because the entire process has been automated. With the discipline of more testing more frequently, we can iterate faster with immediate feedback on the impact of changes. This enables the team to drive quality code with a high assurance of stability and security. Developers will know through immediate feedback whether the new code works and whether any breaking changes or bugs were introduced. Mistakes caught early on in the development process are the easiest to fix.

Purpose of the document

This section discusses the ways in which you can use the best practices when implementing a CI/CD model and how we used these practices into our project.

Continuous integration, delivery and deployment are software development practices born out of the DevOps movement. They make the process of building, testing and releasing software more efficient and get working software into the hands of users more quickly than traditional methods. Done well, a CI/CD pipeline enables teams to deliver working software at pace and get timely feedback on their latest changes.

Each commit triggers a set of automated tests to provide prompt feedback on the change. In order to reap the benefits of continuous integration, it's essential for everyone to share their changes with the rest of your team by pushing to main (master) and to update their working copy in order to receive everyone else's changes. As a general rule aim to commit to main (master) at least once a day.

Security first approach, Strong privilege management is a necessity in fast-moving application pipelines. Integration with secrets management tools and a granular IAM policy engine like AWS IAM is crucial, along with integrations with directory services. Security teams helps to define the appropriate least privilege access models needed for all stages of application development and deployment, and then implement this in a centralized tool/service whenever possible.

Scope of the document

In this section we want to highlight the key points we considered when developing this system and how we achieve each one of them:

Highly Availability is the ability of a system to operate continuously without failing for a designated period. Availability can be measured relative to a system being 100% operational or never failing -- meaning it has no outages. In order to guarantee the high availability for our system we created to elastic containers, this way when we are running a updated in one of the containers the other will still be working, not affecting the end-user usability.

Scalable: A scalable cloud architecture is made possible through virtualization. Different then a physical machine where resources and performance are relatively set, virtual machines (VMs) are highly flexible and can be easily scaled up or down. They can be moved to a different server or hosted on multiple servers at once; workloads and applications can be shifted to larger VMs as needed. When using CodePipeline, each pipeline run executes independently and in parallel with

each other. Since CodePipeline is a managed service, we can run many pipelines in parallel without having to deal with additional infrastructure.

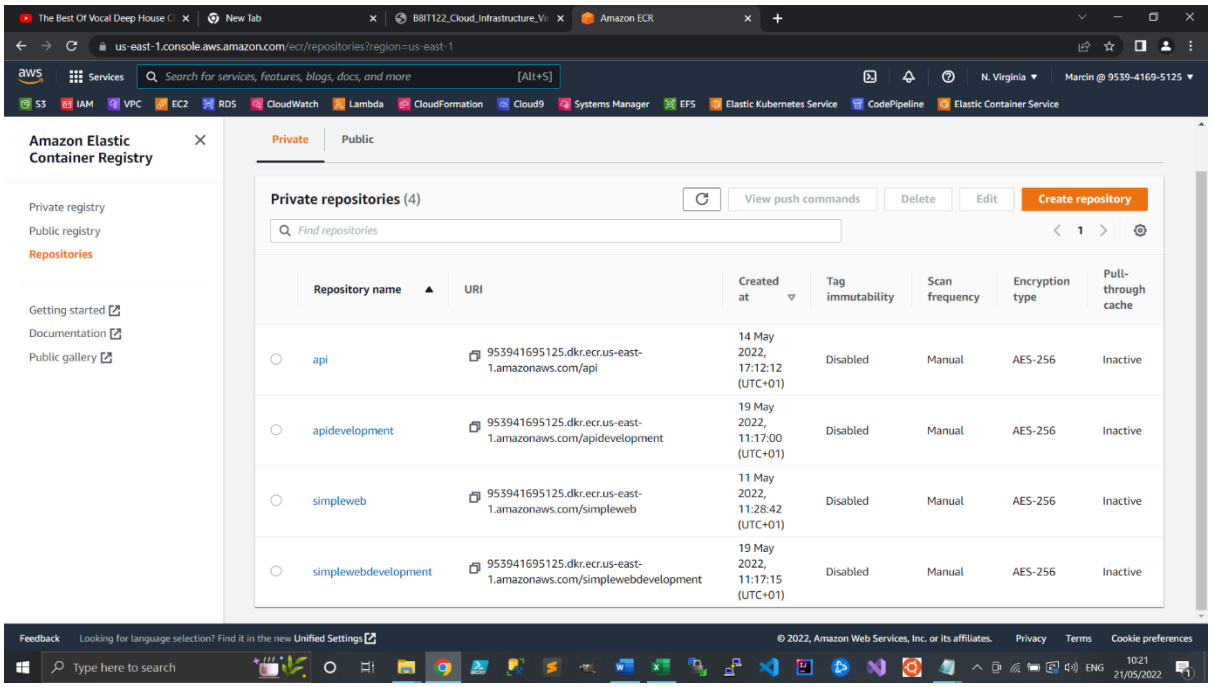
For this project we used two different Availability Zones, in distinct locations within an AWS Region. They provide inexpensive, low-latency network connectivity to other Availability Zones in the same AWS Region.

and Secure System: Cluster is deployed in private subnet 1 and private subnet 2, DB is receiving traffic just from security group of hypervisors and Cloud9 instance. Containers receive traffic just from load balancer security group and DB security group. Load balancer security group accept traffic just from ports that are used by project. Nothing else is open.

Environment information

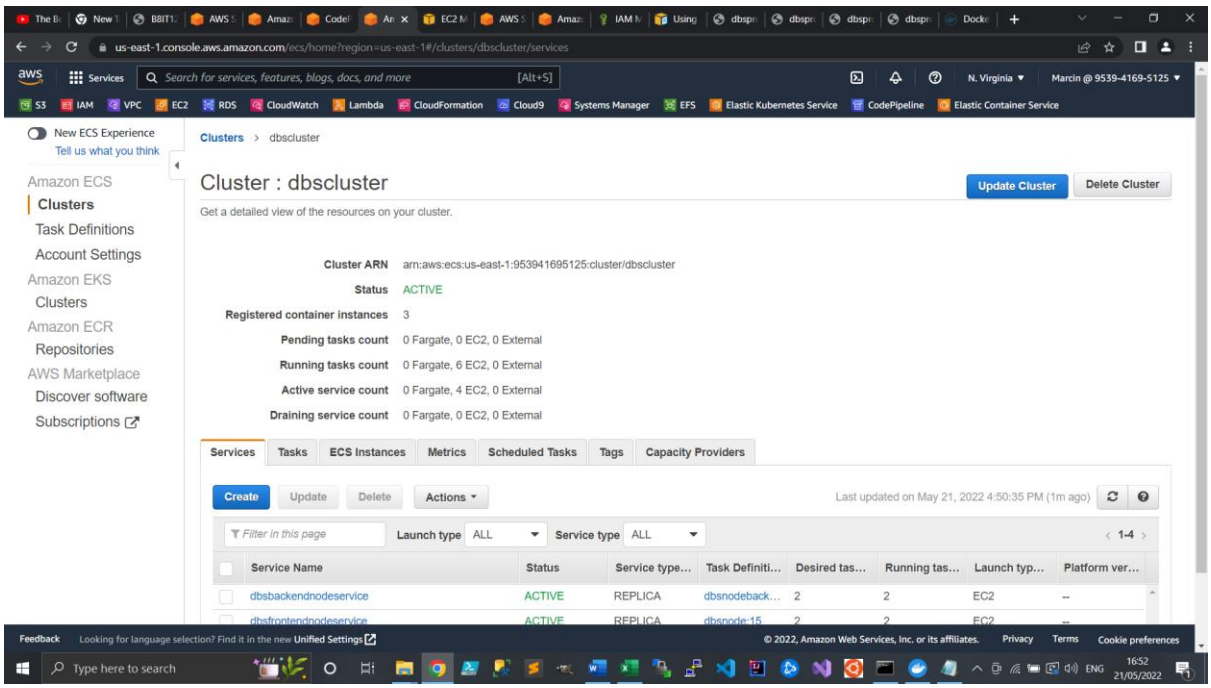
Below you'll find the tools we used to achieve the requirements:

ECR – Amazon Elastic Container Registry - Containers Repository to hold our containers that are built by CodeBuild. ECR is a fully managed container registry offering high-performance hosting, so you can reliably deploy application images and artifacts anywhere. We have 4 container repositories created for project:

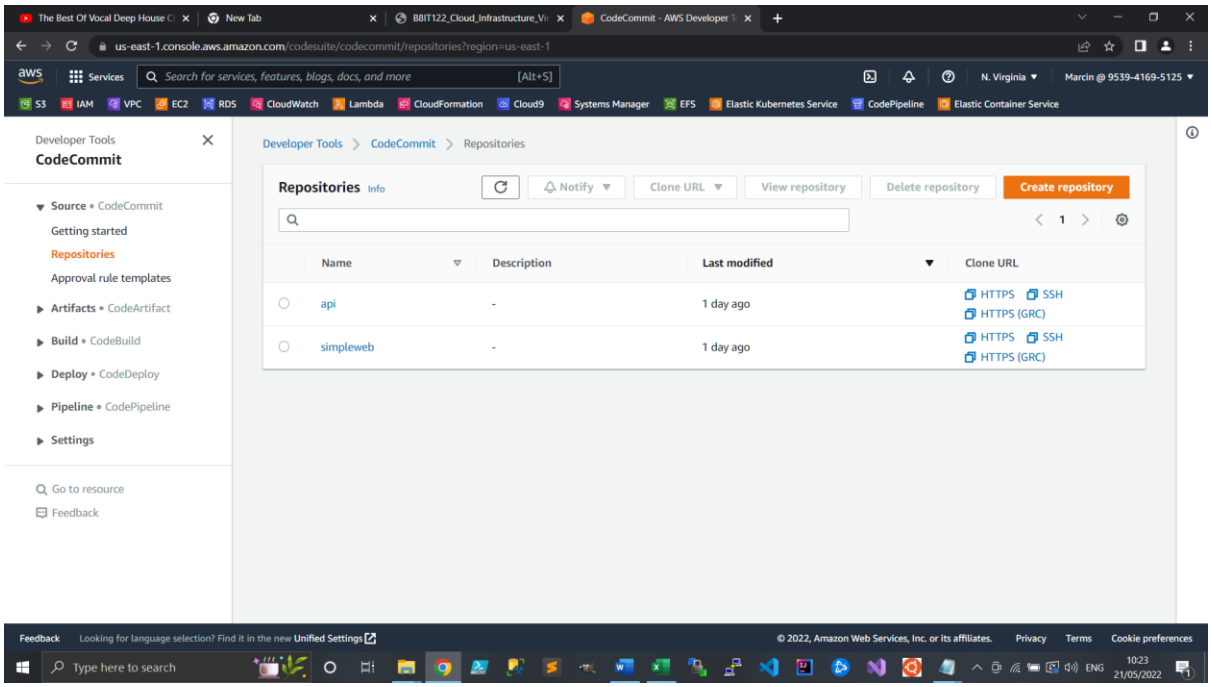


ECS – Amazon Elastic Container Service (Amazon ECS) is a highly scalable and fast container orchestration service. We used it to run, stop, and manage containers on a cluster. With Amazon ECS, our containers are defined in a task definition that we use to run an individual task or task within a service. In this context, a service is a configuration that you can use to run and maintain a specified number of tasks simultaneously in a cluster. We can run your tasks and services on a

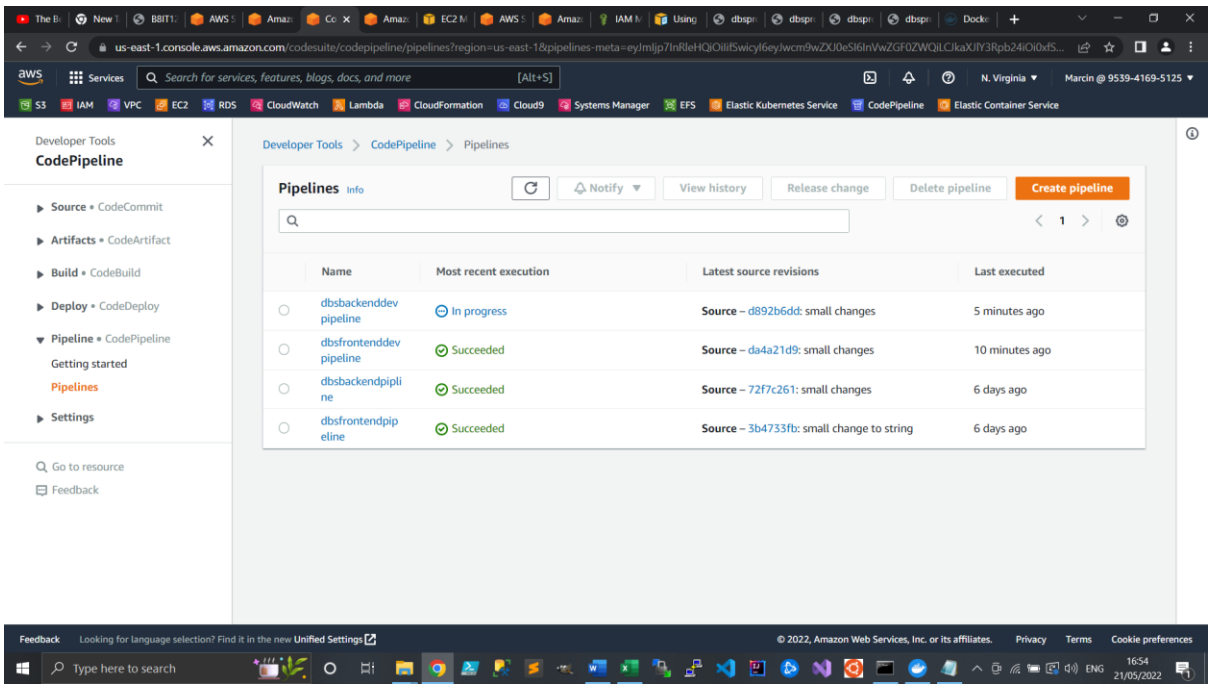
serverless infrastructure that's managed by AWS Fargate. Alternatively, for more control over your infrastructure, we can run your tasks and services on a cluster of Amazon EC2 instances that we manage. For propose of this project we run ECS on EC2.



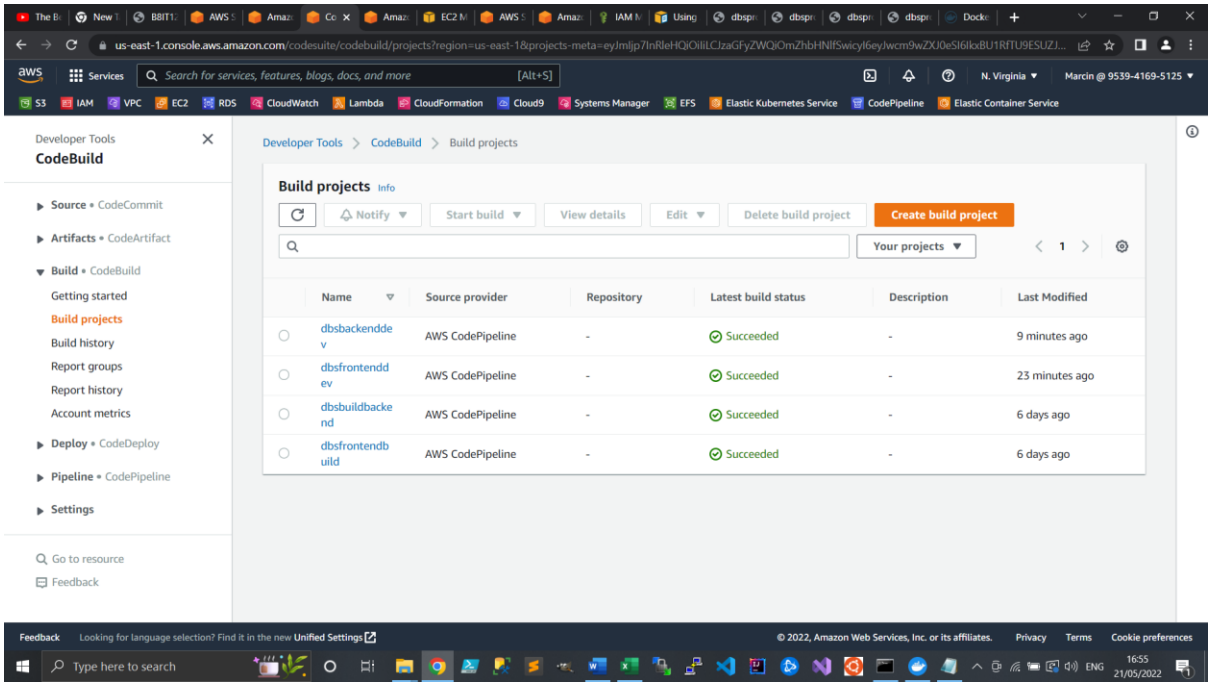
CodeCommit – AWS Code Repository. CodeCommit is a secure, managed source control service that hosts our repositories. We have created 2 repositories with 2 branches each main and development:



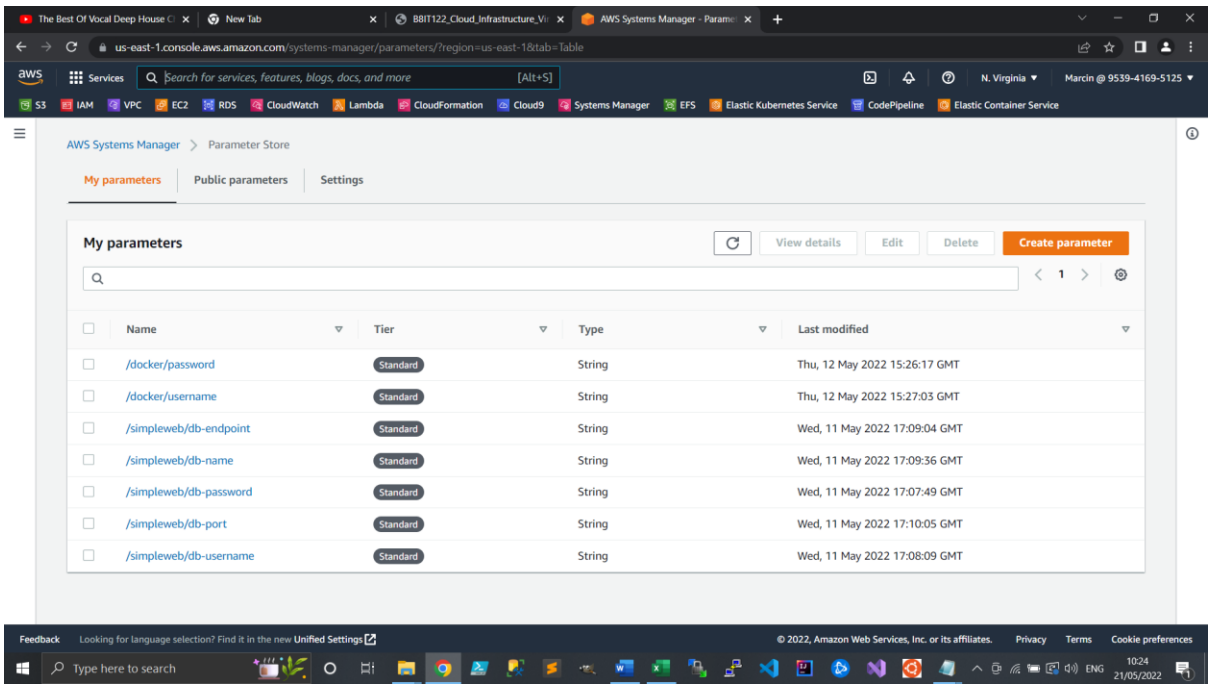
CodePipeline – AWS Service to Build CI/CD. Fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. We build 4 pipelines: frontend, backend, frontend development and backend development.



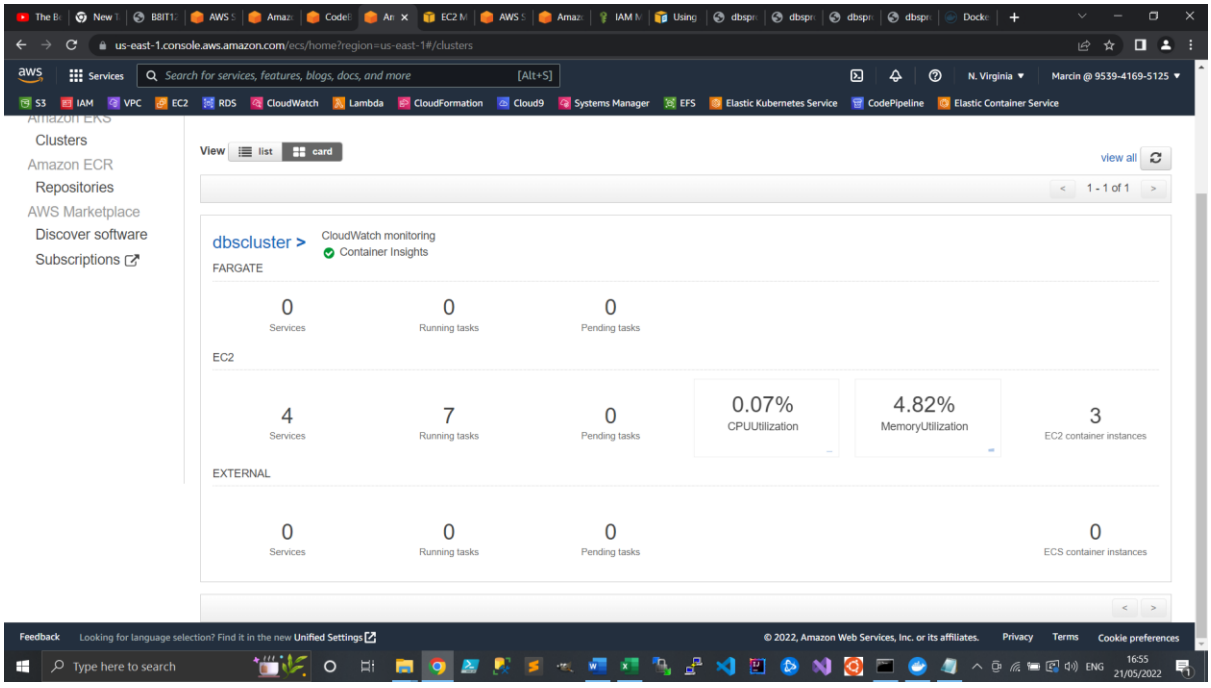
CodeBuild – AWS Service that compiles source code runs tests and produces software packages to deploy. For our pipeline we have 4 build projects:



Parameter store – AWS System Manager parameter store for holding secrets. We used this service to store data such as passwords, database strings, Amazon Machine Image (AMI) IDs, and license codes as parameter values.

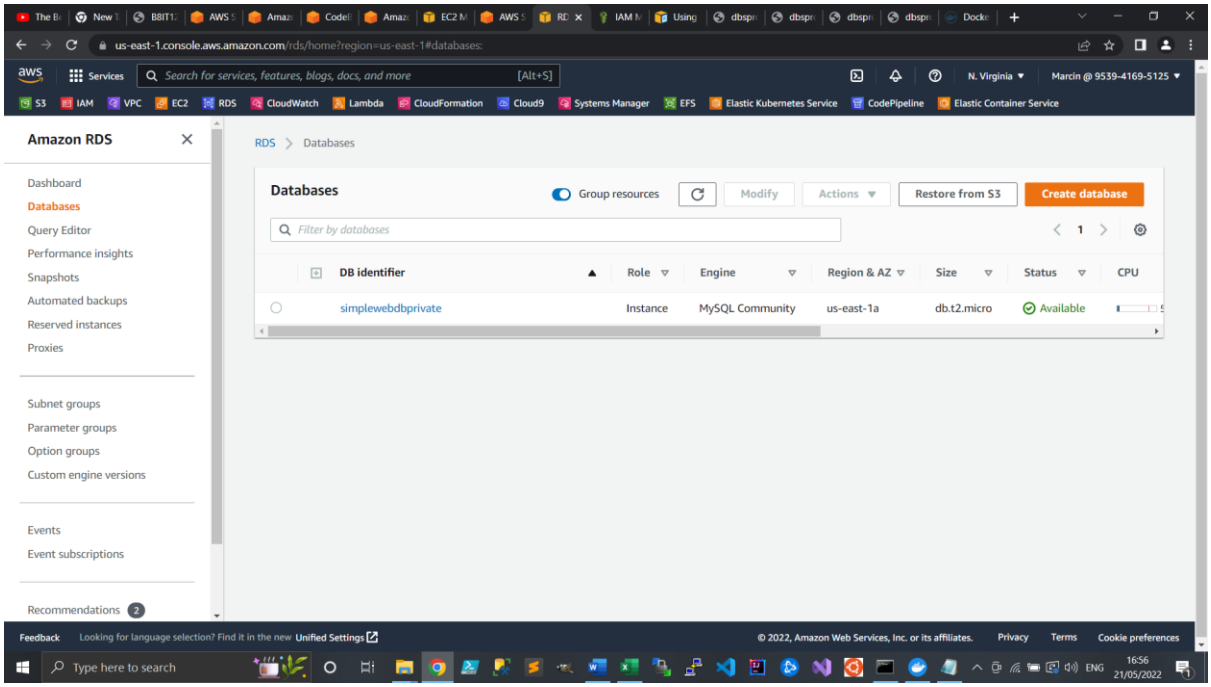


CloudWatch event – AWS CloudWatch logs and events. This service describes changes in Amazon Web Services (AWS) resources, also provide for logs.



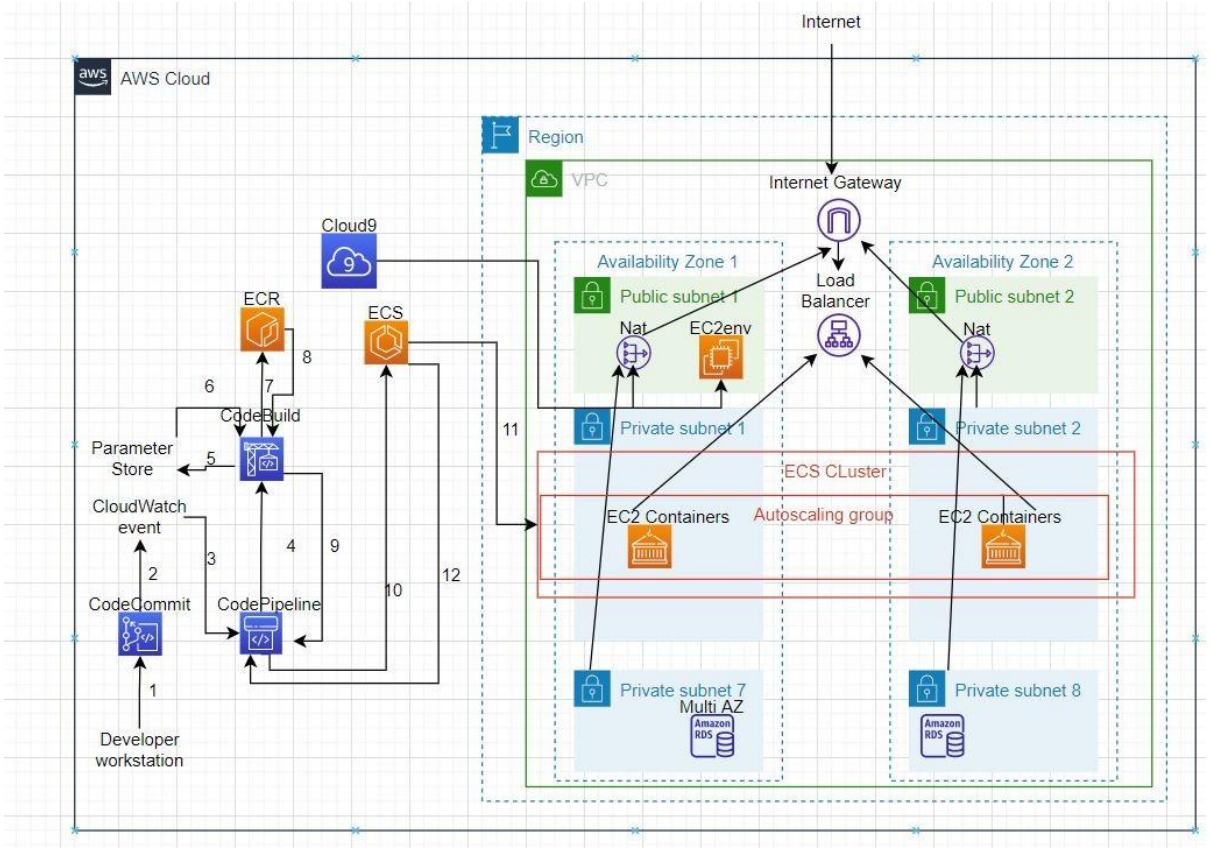
Cloud9 – AWS Cloud IDE. Cloud-based integrated development environment (IDE) that users use to write, run, and debug code or connect to DB

RDS – AWS Service for DB



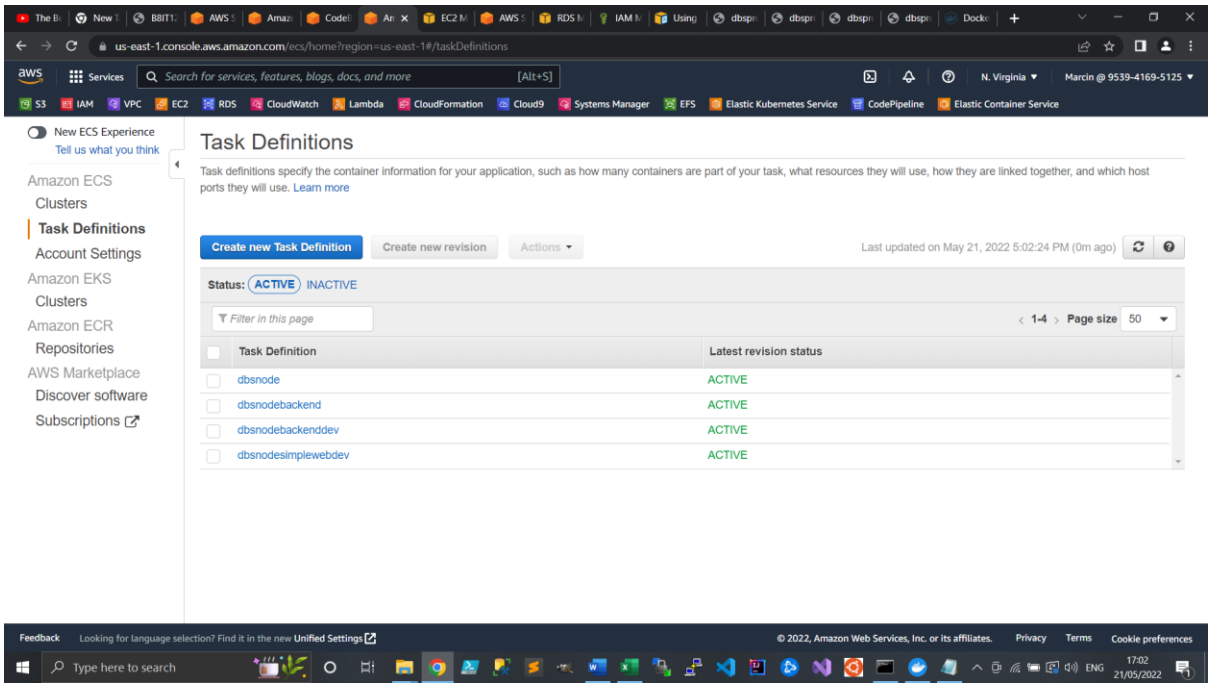
Architecture

Design Specification

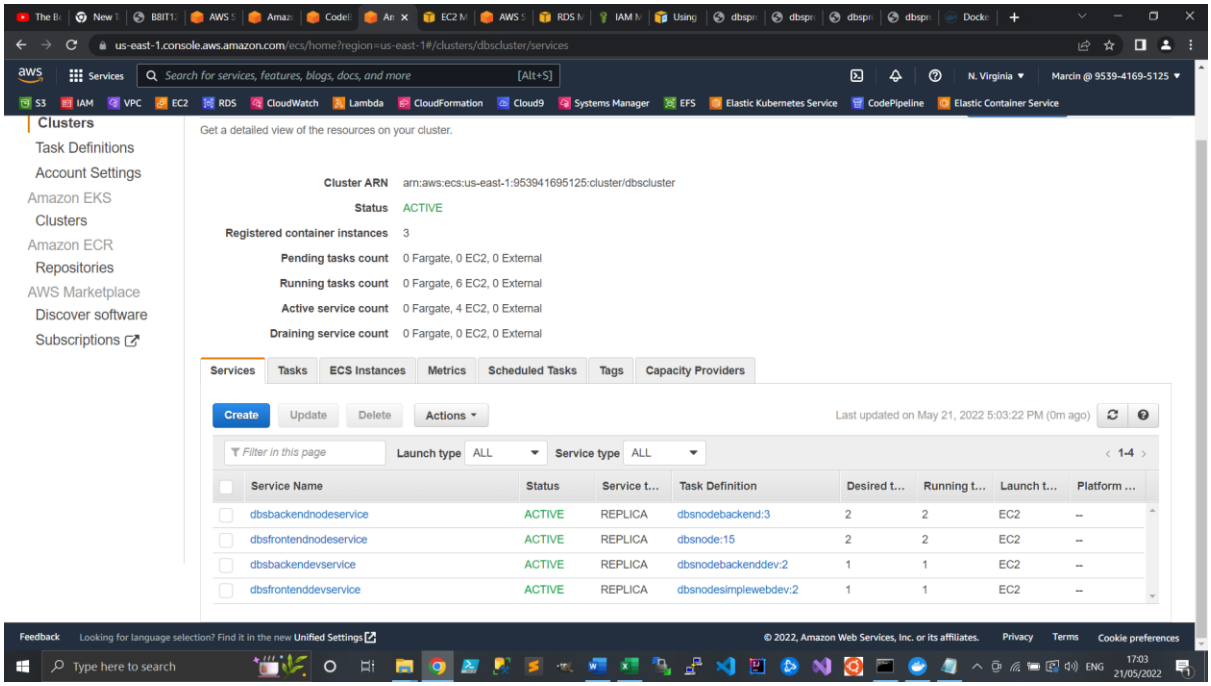


Our ECS containers cluster is deployed on Private subnet 1 and Private subnet 2 on EC2's. Containers accept traffic just from security groups of load balancer and DB MSQL which is on AWS RDS. Containers outbound traffic is going through NAT which is different for Private subnet 1 and Private subnet 2. Autoscaling group deploy 3 EC2 servers for the cluster as desired but up to 5 if needed by ECS cluster which check if CPU usage is above 80% on already running servers. ECS cluster have services which maintain state of containers by running task definition. Service which maintains state of frontend production have min of 2 task running (2 containers) also backend production service have min 2 task running and max 5. Services which maintain frontend and backend development have min 1 task running and max 2. Deployment is set as spread AZ for production and binpack for development. For propose of this project we have created 4 different task definitions and 4 services:

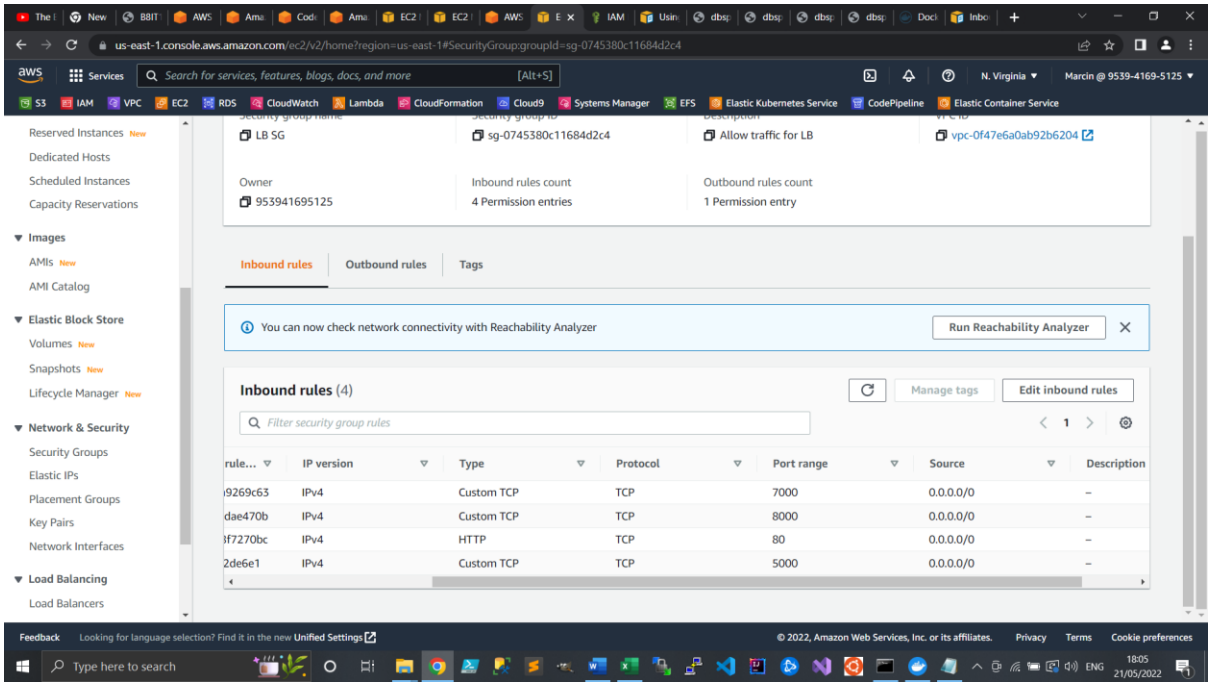
Task Definitions



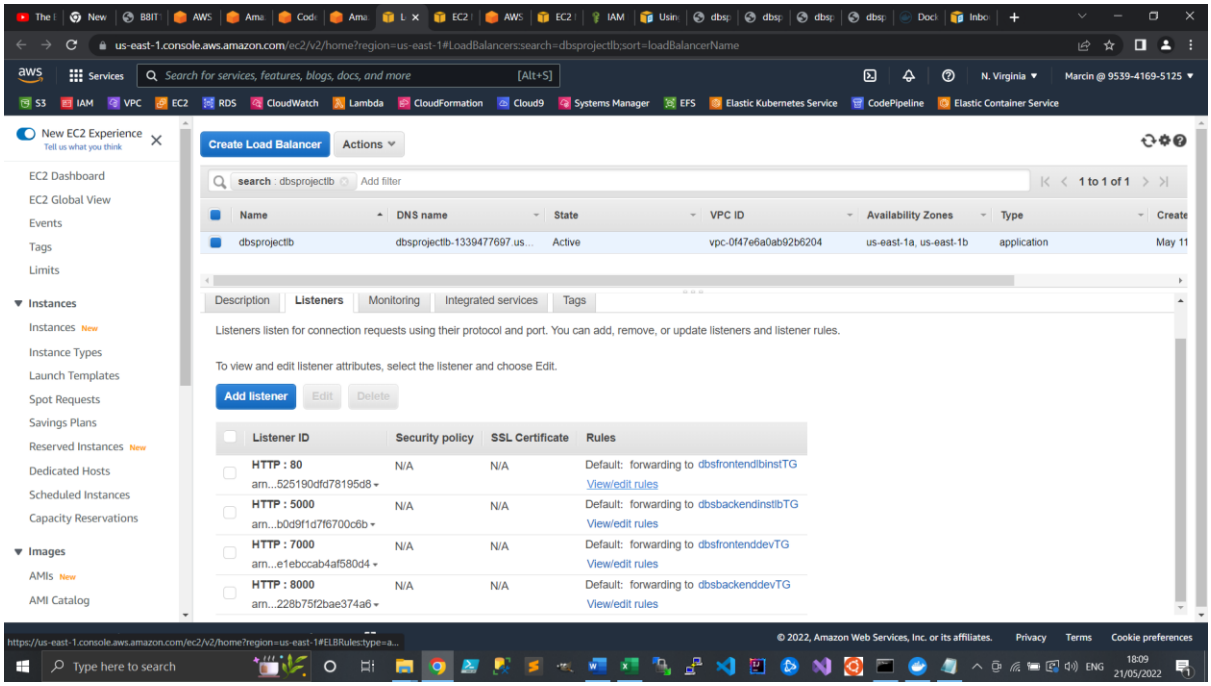
Services



Load balancer accept traffic just needed by our apps:



Have 4 listeners and 4 target groups:



Cloud9 Instance accept traffic just from DB security group and needed by Cloud9 service, users can easily access DB from IDE.

CI/CD steps explained:

2 pipelines were developed: Frontend and Backend, both using the tools listed below:

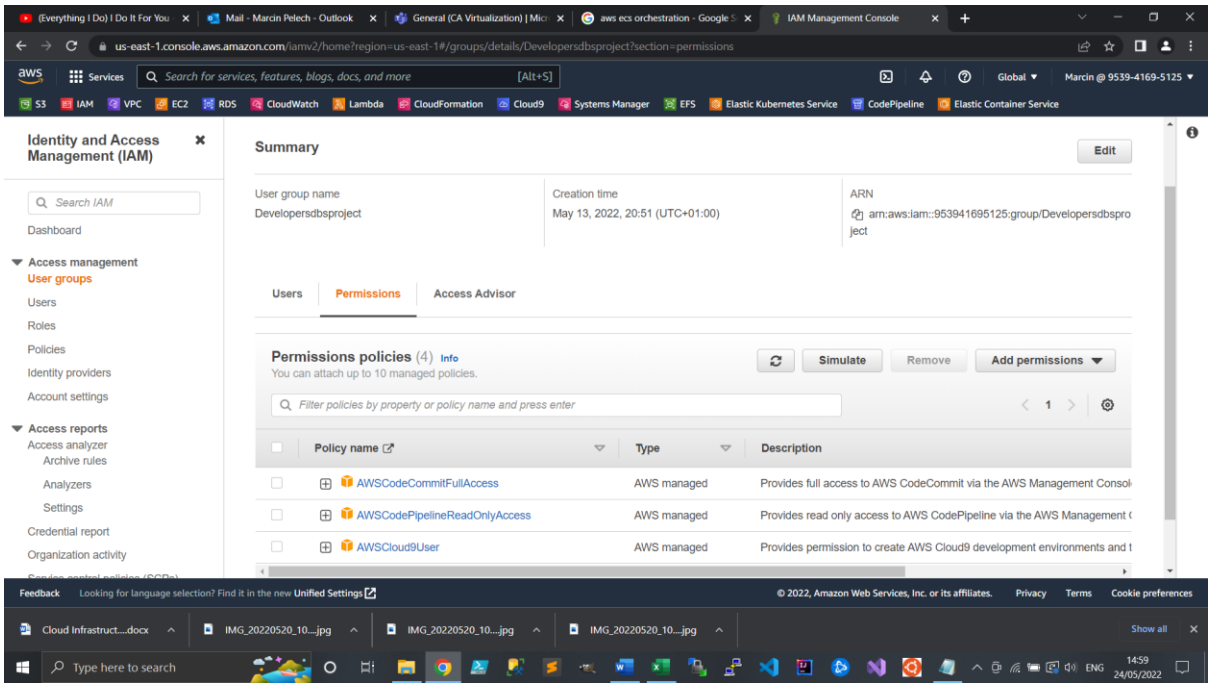
1. Developer ran git push command to CodeCommit successfully
2. CodeCommit received update successfully
3. CloudWatch see event and is sending information to CodePipline
4. CodePipeline start pipeline for which event was triggered, running test on source which is CodeCommit if results are positive going to next step
5. CodeBuild project is starting building docker image, service is gathering necessary information from parameter store
6. CodeBuild Project received necessary details from parameter store and is continue building docker image
7. Once container is ready. CodeBuild is sending docker image to ECR repository
8. uploaded to ECR is successful
9. CodeBuild is informing pipeline of project completion successfully
10. CodePipeline pipeline is sending information for deployment to ECS

11. ECS is sending information do cluster to start deployment
12. ECS is sending information to pipeline of deployment completion

Access Procedure

In order to have a more secure environment, group role was created:

For the developers group role was created: Giving full access to CodeCommit repository, Read-only access to CodePipline and access to Cloud9 as user from where they can connect to database



IAM Groups

IAM Groups are a way of grouping IAM users and IAM roles. Permissions given to an IAM Group are passed onto their group members (users and roles).

The IAM permissions given to an IAM Group (or IAM user or IAM role) determine which AWS API commands can be executed using the AWS CLI or any of the many AWS SDKs.

Resources:

https://www.ukessays.com/essays/computer-science/ci-and-cd-in-aws-code-pipeline.php

https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/practicing-continuous-integration-continuous-delivery.pdf

https://aws.amazon.com/ecr/

https://mng.workshop.aws/ssm/capability_hands-on_labs/parameterstore.html

https://www.techtarget.com/searchdatacenter/definition/high-availability

https://www.vmware.com/topics/glossary/content/cloud-scalability.html#:~:text=A%20scalable%20cloud%20architecture%20is,easily%20scaled%20up%20or%20down.

https://www.jetbrains.com/teamcity/ci-cd-guide/ci-cd-best-practices/