



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Life Cycle Management of Serverless Microservices using  
Amazon Web Services

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Yatsenko , Oleksandr

Tutor/a: Moltó Martínez, Germán

CURSO ACADÉMICO: 2022/2023

## Resumen

---

El presente Trabajo Fin de Grado consiste en el análisis y creación de aplicaciones basadas en microservicios utilizando técnicas de “serverless computing”. Como una plataforma en la nube se va a utilizar el proveedor de Cloud público Amazon Web Services y sus servicios. Para demostrar la parte práctica y aplicar todas las técnicas se va a usar una aplicación Serverless de ejemplo “Airline Booking”. Se analizarán las estrategias necesarias para el Despliegue, Escalado, Actualización, Visibilidad/Traceo y Securización.

**Palabras clave:** Microservicios, AWS, AWS Lambda, AWS X-RAY, Cognito, Blue/Green despliegue, Canary release, AWS SAM.

## Abstract

---

This Final Project consists of analysis and creation of microservices using "serverless computing" techniques. As a cloud platform, the Amazon Web Services public Cloud provider and its tools will be used. To demonstrate the practical part and apply all the techniques, the exemplary Serverless application "Airline Booking" will be used. The necessary strategies for Deployment, Scaling, Update, Visibility / Tracing and Securing will be analyzed.

**Keywords:** Microservices, AWS, AWS Lambda, AWS X-RAY, Cognito, Blue/Green deploy, Canary release, AWS SAM.



# Table of Contents

---

Chapter 1. Introduction	5
1. Motivation	6
2. Objectives	8
3. Structure of Document	8
Chapter 2. Related Technologies and Tools	9
1. Deployment. AWS SAM & Serverless framework	9
2. Scaling. AWS Lambda	11
3. Updating. Canary release and blue-green deployment	12
4. Visibility and Traceability. AWS X-Ray	17
5. Securing. Amazon Cognito	19
6. Tools	21
Chapter 3. Exemplary Microservices-based Application	27
1. Application architecture	27
2. Project structure	28
Chapter 4. Deployment	31
1. Deploy	31
2. Scaling. Lambda functions	35
3. Cognito	44
5. AWS X-RAY	53
Chapter 5. Conclusions and Future Work	57
Bibliography	59
Appendix. SDG	62



Figure 1 AWS Lambda Diagram	12
Figure 2 Canary release	14
Figure 3 Blue-green deployment	15
Figure 4 X-RAY	18
Figure 5 Amazon Cognito	20
Figure 6 Free tier account	22
Figure 7 GitHub page	23
Figure 8 Visual studio code	24
Figure 9 NodeJs & npm	26
Figure 10 Application structure	28
Figure 11 Application structure	29
Figure 12 Lambda functions	30
Figure 13 Amplify hosting	32
Figure 14 Permissions policies	32
Figure 15 Deployed application	32
Figure 16 Deployed application on AWS	33
Figure 17 IAM user	34
Figure 18 Amplify initialization	34
Figure 19 Amplify result	35
Figure 20 Backend environment	35
Figure 21 Provisioned Concurrency	36
Figure 22 Scale-up code for Lambda	39
Figure 23 Succeeded test Scale-up function	39
Figure 24 Scale-down code for Lambda	40
Figure 25 Succeeded test Scale-down function	41
Figure 26 Monitor resource usage code for Lambda	43
Figure 27 Succeeded test Monitor resource usage function	43
Figure 28 Access AWS services with a user pool and an identity pool	44
Figure 29 New user pool	46
Figure 30 Identity pool	46
Figure 31 Canary creation	47
Figure 32 Canary monitoring	48
Figure 33 Canary runs	49
Figure 34 IAM role for Blue/Green deployment	49
Figure 35 ELS	50
Figure 36 Launch configuration	50
Figure 37 Auto Scaling group	50
Figure 38 CodeDeploy creation	50
Figure 39 Deployment group creation	51
Figure 40 New deploy on CodeDeploy	51
Figure 41 Active instances in CodeDeploy	51
Figure 42 Blue environment	52
Figure 43 Green environment before deployment	52
Figure 44 Green environment after	52
Figure 45 X-RAY traces option	53
Figure 46 X-RAY SDK installation	54
Figure 47 X-RAY implementation	55
Figure 48 X-RAY traces	55

# Chapter 1. Introduction

---

Nowadays, computer science is the fastest growing industry and one of the highest paid. Analysts, frontend & backend developers, database administrators - these and dozens of other specialities in the field of IT technologies are at the peak of demand. It is applicable in almost all areas of life, such as Medicine, Engineering, Business, Military, Government and others. Therefore, every year it expands and gains popularity in society. With this quite high demand, the number of new tools, libraries, programming languages and cutting-edge solutions are also growing, making the development and creation of new products easier and more accessible for everyone. As a result, the field continues to expand, gaining popularity and shaping the future. The tech industry is constantly evolving, introducing new technologies that streamline the development process and make it more accessible to a wider range of individuals.

Cloud solutions have played a significant role in this democratization process. With the advent of cloud computing, developers now have access to scalable and cost-effective infrastructure without the need for significant upfront investments in hardware and data centers. In recent years, serverless techniques [1] have emerged as a game-changer in the world of cloud computing. Serverless architectures abstract away the underlying infrastructure, allowing developers to focus solely on writing code. In a serverless environment, applications are broken down into smaller functions that are executed in response to specific events. This event-driven paradigm not only simplifies development but also enhances scalability and cost efficiency.

First of all, let's give a definition of Microservices and Serverless computing [2]. Microservices is an approach in which a system is built as a set of independent and loosely coupled services that can be created using various programming languages and data storage technologies. Serverless computing is a way to provide server services without renting/purchasing physical equipment. Servers in this case, of course, are used, but on the side of the service provider. The user does not interact with the infrastructure in any way and does not serve it, but at the same time he can write and deploy code using ready-made computing resources. To do this, it will focus on the fundamental characteristics of microservices-based application architectures and will determine those strategies, techniques, and tools that can be applied in the context of the AWS Lambda service [3], an example of a serverless service.

Serverless architectures are event-driven and highly scalable. Developers can focus on writing code in the form of functions or microservices, which are executed in response to specific events or triggers. These events can include HTTP requests, database changes, file uploads, or scheduled tasks. With serverless, the infrastructure is abstracted away, and developers can concentrate on writing business logic rather than managing servers.



One of the key benefits of serverless architectures is their ability to scale automatically based on demand. Cloud providers, such as AWS Lambda, dynamically allocate resources to handle incoming requests. This auto-scaling feature ensures optimal performance and cost efficiency, as resources are provisioned only when needed. Organizations can handle high traffic loads without worrying about infrastructure provisioning or capacity planning. Serverless architectures also offer built-in fault tolerance and high availability. Cloud providers handle underlying infrastructure failures, ensuring that functions or services are automatically re-executed in case of any issues. This inherent resilience improves application reliability and reduces the impact of potential failures.

### **1. Motivation**

Serverless architectures and microservices offer several benefits, including reduced operational overhead, improved scalability and cost-effectiveness, and faster time-to-market for new applications and features. However, managing the life cycle of serverless microservices can be complex and challenging, particularly in large and dynamic environments. This is where Amazon Web Services (AWS) comes into play, providing a range of tools and services to help manage the life cycle of serverless microservices, including AWS Lambda, AWS IAM [4], AWS SAM [5] and others.

The purpose of this final degree project is to explore some of the various tools and services offered by AWS for life cycle management of serverless microservices. The project aims to identify the key challenges and best practices for managing the life cycle of serverless microservices in AWS, and to demonstrate how these tools and services can be used to build, deploy, and manage scalable and highly available microservices. The project also explores topics such as continuous integration and continuous delivery (CI/CD), monitoring and logging, security and compliance. The project involves a hands-on approach, using real-world examples and scenarios to demonstrate the concepts and techniques learned. In conclusion, the motivation for this final degree project is to explore the challenges and best practices for life cycle management of serverless microservices in AWS, and to demonstrate how the tools and services offered by AWS can be used to build, deploy, and manage scalable and highly available microservices.

In addition to creating a new product, service, website, or a simple application, it needs to be published, the data should be stored, secured and have full access to it. One of the popular solutions are web hosting, server rental or creating your own server room. If this is a small application or a medium-sized service with a database, then maintaining your server can be quite expensive due to the purchase of the necessary hardware, maintenance cost and hiring staff for its correct operation 24/7. It is necessary to calculate and predict the load correctly so as not to overpay for downtime in standby mode, or vice versa. If the load is not estimated well, then it is possible to lose potential customers, clients, sales, and this will certainly have a bad effect on the company / product / service. Therefore, every day, the topic of cloud storage is gaining great

demand among small and medium-sized businesses. Even tech giants such as Netflix, Adobe, Airbnb, Slack, Twitch, Spotify, etc. are using cloud solutions. For example:

- Netflix as one of the largest video streaming platforms utilizes AWS services including Amazon EC2 for computing, Amazon S3 for storage, Amazon RDS for databases, and Amazon CloudFront for content delivery
- Adobe utilizes AWS to power its cloud-based services. Adobe Creative Cloud and Adobe Experience Cloud are hosted on AWS, providing users with access to their creative tools and marketing solutions.
- Airbnb utilizes AWS for its scalability and reliability. AWS services like Amazon EC2, Amazon S3, and Amazon RDS are key components of Airbnb's infrastructure, allowing them to handle millions of bookings and user interactions.
- Slack relies on AWS to power its messaging platform. Amazon EC2, Amazon S3, Amazon RDS, and Amazon CloudFront provide the scalability and performance required to handle millions of concurrent users.
- Twitch is built on AWS infrastructure. Twitch uses AWS services like Amazon EC2, Amazon S3, Amazon DynamoDB, and Amazon CloudFront to support live video streaming, chat functionality, and community
- Spotify uses services like Amazon EC2, Amazon S3, and Amazon CloudFront helps Spotify handle the storage, delivery, and scalability requirements of its vast music catalog.

Thanks to such a service you do not have to worry about buying and maintaining your own servers, and flexible configuration will allow customizing it to the needs of the client, depending on the required power, traffic volume, data volume, cost. As a result you can get a stable working product (cloud services guarantee from 99% uninterrupted operation and compensation in case of a non-working service on their part), flexible pricing policy (pay for what is used) and different settings for different scenarios, such as:

- increase/decrease requests to the server
- data recovery
- global access to the data
- cyberattacks

To demonstrate the power of microservices and serverless architecture Amazon Web Services will be used as the cloud computing platform and the Airline booking Application that is provided by AWS as an example. For deployment, storing data, updates, and maintenance AWS's products would be used, such as: AppSync, Amazon API Gateway, Lambda, Amplify [6], X-RAY [7], DynamoDB and Cognito [8] (all the product's definitions would be described below). This platform had been chosen because it has a free tier account for testing all of their products with certain limits.





AWS appeared on the market in 2006 and today occupies one of the leading places in terms of the provided capacities. Its services are used in various industries, such as: Advertisement & Marketing, Financial Services, Healthcare, Retail, Education, Energy, Government, etc.

## 2. Objectives

The main goal of this project is to explain what microservices are, how they work, how this cloud solution can be implemented to the typical application and show how a real project could be adapted to this approach, and showcase an example of an existing application.

This work will address the process of creating microservices using serverless computing techniques exemplified on AWS Lambda to manage the computation, and API Gateway to offer the REST API after which the service is exposed. Among others, the strategies, and tools necessary for the following characteristics of microservices will be analyzed:

- 1) Deployment, covering tools like AWS Amplify and the Serverless framework [1].
- 2) Scaling, using provisioned concurrency techniques, and performance impact of using it.
- 3) Updating, using canary release [9] and blue-green deployment [10] techniques for changing between software versions.
- 4) Visibility and traceability, using telemetry tools such as AWS X-Ray to analyze and debug distributed applications.
- 5) Securing, through the integration of services such as Amazon Cognito.

## 3. Structure of Document

This document is composed of five sections:

1. Introduction. The motivation and objectives of the project are presented. This section describes the purpose and the goals of this project
2. State of the art. This section describes the products and techniques that would be used or represented in this work. The main goal is to have a global vision of the project and how each part is working.
3. Analysis. This section explains each technology, service or platform that is used in this project.
4. Implementation. This section shows how all these techniques and products work together as a whole. It describes deeply the way how these tools and products are implemented and customized.
5. Conclusions. This section is about the achievement of objectives that are defined in the introduction section and future work.

# Chapter 2. Related Technologies and Tools

---

This part briefly describes all the sections that are implemented in this project, which instruments/technologies/frameworks were used, their advantages and description. It gives a short theoretical explanation and understanding that it is needed for the practical part.

## 1. Deployment. AWS SAM & Serverless framework

The Serverless framework is one of the most well-known and popular frameworks for building serverless applications. Due to its compatibility with several backend options, it is widely used in conjunction with other frameworks. Users may choose their preferred platform among Microsoft Azure Functions, Google Cloud, and AWS thanks to the Serverless Framework.

### Advantages:

- Open-source framework;
- Supports functions written in different languages, including Python, C#, F#, Go, Node.js, and Ruby;
- Extensible with plugins;
- Supports different backends, including Kubernetes;
- Has a flexible variable system;
- Supports many cloud platforms simultaneously;
- It updates AWS lambda functions and associated triggers using a straightforward YAML abstract syntax.

**AWS SAM**, also referred to as the AWS Serverless Application Model, serves as a powerful open source platform that empowers users to effortlessly construct serverless applications within the AWS ecosystem. This cutting-edge platform offers a comprehensive template specification, enabling users to precisely define their serverless applications, while the command line interface (CLI) tool provides a seamless and efficient development experience. Additionally, AWS SAM expands the functionality of Amazon Web Services CloudFormation by presenting a streamlined approach for identifying the essential AWS Lambda, Amazon DynamoDB, and Amazon API Gateway APIs that are crucial for the optimal performance of your serverless application.

Here are several notable benefits associated with leveraging AWS SAM for serverless application development:



- Streamlined deployment: AWS SAM simplifies the deployment process through its template-based syntax, enabling developers to define and deploy serverless applications effortlessly. This facilitates a smooth onboarding experience and instills confidence in application deployment.
- Enhanced resource management: With AWS SAM, developers can utilize a concise and intuitive syntax to define AWS resources, including AWS Lambda functions, Amazon API Gateway APIs, and Amazon DynamoDB tables. This streamlined approach empowers developers to efficiently manage their resources and optimize application performance.
- Improved resource organization: AWS SAM introduces a hierarchical structure for organizing AWS resources, providing developers with a clear framework to manage application components and establish meaningful relationships between various resources. This organizational paradigm enhances overall development efficiency and resource management.
- Robust testing and debugging: AWS SAM equips developers with a local testing environment, enabling them to rigorously test applications locally before deploying them to the cloud. This facilitates comprehensive debugging and issue resolution, minimizing the impact on production environments.
- Strengthened security: AWS SAM incorporates security best practices and guidelines specifically tailored for deploying secure serverless applications. This empowers developers to build and deploy applications that meet the stringent security requirements of their organizations, fostering robust and reliable security measures.
- Cost optimization: By leveraging serverless architectures facilitated by AWS SAM, developers can take advantage of a pay-as-you-go model, where costs are incurred only for actual compute resource usage, rather than continuously paying for idle resources. This can lead to substantial cost savings compared to traditional, always-on compute architectures.

In summary, AWS SAM offers a range of advantages including simplified deployment, improved resource management, enhanced resource organization, efficient testing and debugging, strengthened security measures, and potential cost savings. These benefits collectively contribute to a streamlined and efficient serverless application development process.

Apart from AWS SAM, the Amplify service is also required to deploy the application into the cloud. This is a powerful development framework provided by Amazon Web Services that simplifies the process of building scalable and secure web and mobile applications. Here are the features that this service has:

- AWS Amplify streamlines the application development process by offering a simplified workflow and pre-configured backend services. Developers can easily set up authentication, data storage, APIs, and other essential services through a command-line or the AWS Amplify Console.

- It seamlessly integrates with popular frontend frameworks such as React, React Native, Angular, and Vue.js. It provides libraries and UI components that facilitate seamless connectivity between frontend applications and backend services like AWS AppSync for real-time data synchronization or AWS Lambda for serverless computing. This integration simplifies development and ensures a cohesive end-to-end experience.
- This service offers a range of scalable backend services to cater to the demands of applications. Services like AWS AppSync for real-time and offline data synchronization, AWS Lambda for serverless computing, Amazon DynamoDB for NoSQL database storage, and Amazon S3 for file storage are readily available.
- Amplify offers a complete CI/CD pipeline for seamless application deployment and hosting. By connecting code repositories to Amplify, it gets a possibility to automate deployments, enabling continuous delivery of updates. Amplify supports various hosting options, including static website hosting with Amazon S3, serverless backends with AWS Lambda, and server-side rendering with AWS AppSync and Amazon CloudFront.

## 2. Scaling. AWS Lambda

**AWS Lambda** is a dedicated tool that helps you activate any code for any application without having to manage it. Any code will be executed based on events that will occur in AWS services. In simple words: you add your script, which should be executed in AWS Lambda, and set a trigger or event that will run this code. There is no need to do anything else, because - administration, code monitoring, security, logs, etc. - will be taken care of by AWS Lambda service (Figure 1).

### Advantages of leveraging AWS Lambda:

- **Cost optimization:** By utilizing AWS Lambda, you benefit from a cost-efficient approach where you pay solely for the compute time you consume. There are no upfront expenses or long-term commitments, and you can effortlessly scale your application to accommodate varying levels of traffic.
- **Enhanced scalability:** AWS Lambda automatically scales your applications based on incoming request traffic, enabling seamless handling of sudden spikes without requiring manual intervention. This ensures your application remains responsive and accessible even during high-demand periods.
- **Language versatility:** AWS Lambda supports multiple programming languages, including Java, Python, Node.js, and more. This flexibility empowers you to develop your application using the language of your preference, facilitating a smoother development experience.
- **Reduced operational burden:** With AWS Lambda, you are relieved of concerns regarding server maintenance, infrastructure provisioning, and capacity planning. This enables you to focus on core application development and delivery while AWS manages the underlying infrastructure.



- **Improved reliability:** AWS Lambda executes your code in a highly available and redundant environment, ensuring continuous application availability. Automatic scaling and failover capabilities minimize downtime and guarantee high levels of reliability.
- **Seamless integration with AWS services:** AWS Lambda seamlessly integrates with numerous other AWS services, such as Amazon S3, Amazon DynamoDB, Amazon SNS, and more. This simplifies the building, deployment, and management of applications that leverage these services.
- **Event-driven computing:** AWS Lambda empowers you to create event-driven applications that respond to specific events, such as modifications to data in an Amazon S3 bucket or the insertion of a new record in a DynamoDB table. This event-driven approach enhances application functionality and responsiveness.
- **In summary,** AWS Lambda offers a flexible, scalable, and cost-effective solution for running applications while minimizing operational overhead and maximizing reliability. Its seamless integration with other AWS services and event-driven computing capabilities makes it an ideal platform for building, deploying, and managing serverless applications.

### Working with AWS Lambda

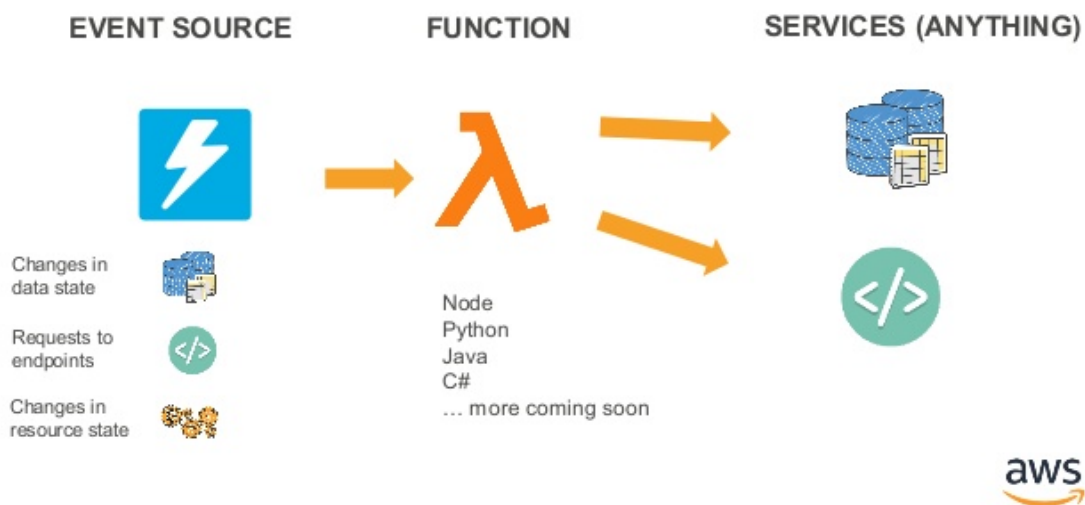


Figure 1 AWS Lambda Diagram [24]

AWS Lambda only invokes code when needed and automatically scales resources to match the volume of incoming requests without any additional action from the client. The number of processed requests is not limited. AWS Lambda typically runs code within a few milliseconds of an event, and because scaling is automatic, function performance remains consistently high as event frequency increases. Because the code runs stateless, Lambda can create as many instances as needed (up to 3000 concurrent invocations) without lengthy deployments or setup delays.

### 3. Updating. Canary release and blue-green deployment

**Canary release** is a strategic deployment method that empowers you to implement changes to your applications gradually and systematically, ensuring a controlled and measurable rollout. By adopting this approach, you significantly mitigate the potential risks associated with introducing new features or updates that may adversely affect the stability and performance of your application.

Within the AWS ecosystem, there are multiple AWS services available to facilitate canary releases, including Amazon Route 53, Amazon CloudFront, and Amazon API Gateway. These services enable you to intelligently direct traffic between different versions of your application as shown in figure 2, ensuring a smooth transition and thorough evaluation of the changes.

The canary release process typically encompasses the following essential steps, ensuring a meticulous approach [11]:

1. Preparation: Careful planning is conducted to define the scope and objectives of the canary release. This involves identifying the specific features or updates to be introduced and determining the metrics and criteria for evaluating the success of the deployment.
2. Multiple versions of the application are created to facilitate a controlled rollout. The canary version, representing the updated changes, is created alongside the existing stable version.
3. Traffic routing is intelligently directed between the canary version and the stable version of the application. This allows for a gradual increase in traffic to the canary version while still serving the majority of users with the stable version.
4. During the canary release, various monitoring tools and techniques are employed to assess the performance, stability, and user experience of the canary version. Metrics and logs are analyzed to detect any anomalies or issues that may arise.
5. Based on the monitoring and evaluation results, the traffic to the canary version is incrementally increased if the performance and stability remain satisfactory. This gradual expansion ensures that any potential issues can be identified early and addressed before a full rollout.
6. Depending on the evaluation results, a decision is made to either rollback the canary version if issues are detected, or to promote it as the stable version if it proves to be successful.

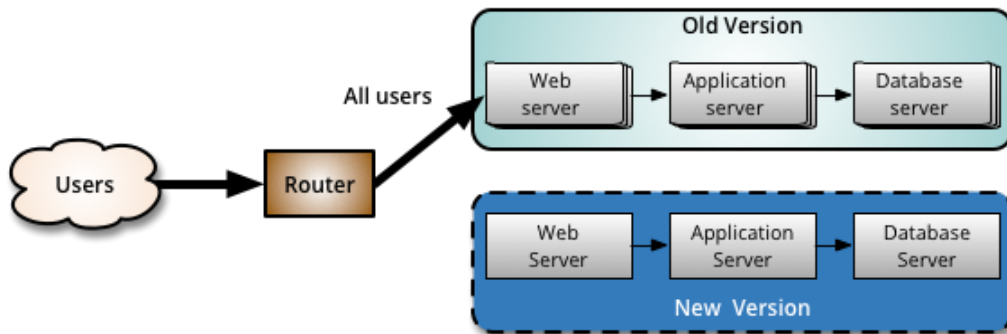
By following this systematic approach, canary releases enable organizations to minimize risks, gather valuable insights, and ensure a smooth transition when introducing changes to their applications.

AWS offers a variety of resources to assist you in putting into practice a canary release, including Amazon CloudWatch for monitoring, AWS X-Ray to analyze and debug distributed applications deployed with canary releases and AWS Lambda for automating deployment and rollback procedures. With the aid of these services, you may create a



solid and dependable canary release procedure that guarantees the dependability and efficiency of your applications.

To sum up, a canary release is an effective deployment approach that enables you to progressively roll out changes to your applications in a controlled and regulated manner. By utilizing AWS services, you may create a solid and dependable canary release procedure that lowers the danger of implementing new features or upgrades that can have an influence on the stability and performance of your application.



*Figure 2 Canary release [25]*

**Blue-green deployment** is a deployment technique that enables you to roll out changes to your application with minimal downtime and risk. This approach allows to test and validate changes in a production-like environment before making them available to the users (Figure 3).

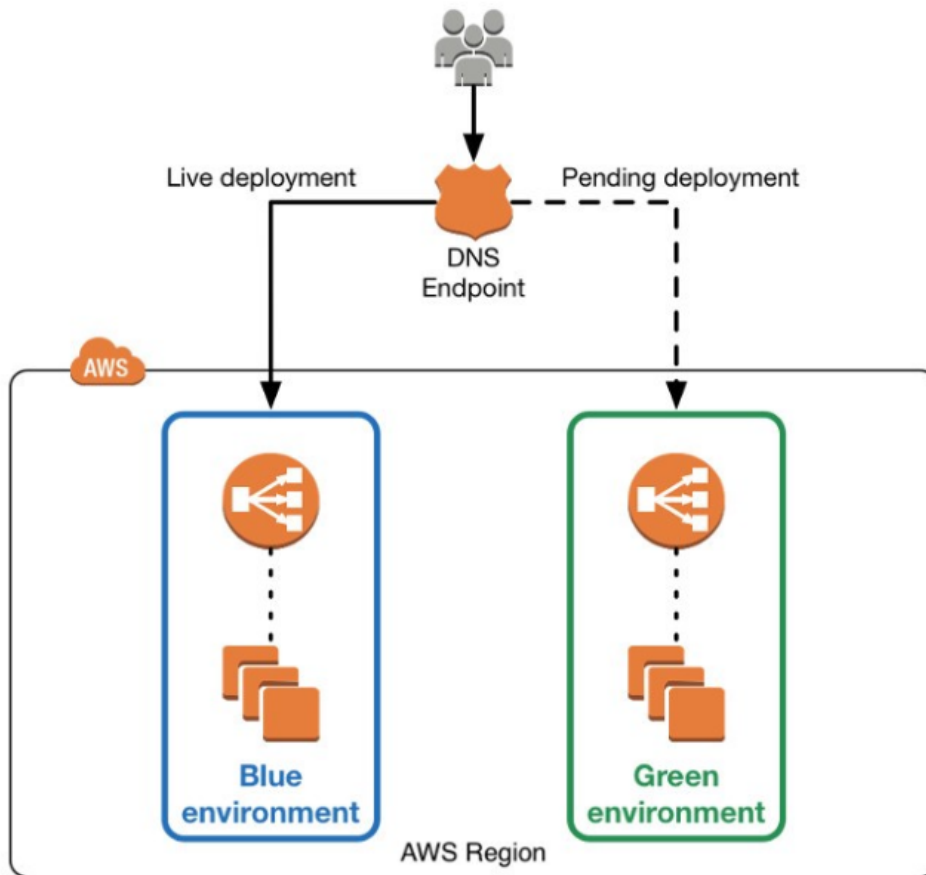


Figure 3 Blue-green deployment [26]

In AWS, a blue-green deployment can be implemented using various AWS services, such as Amazon EC2, Amazon ELB, and Amazon Route 53. The blue-green deployment process typically involves the following steps [12]:

1. Create a new version of your application and deploy it to a separate environment, known as the "green" environment.
2. Verify the new version of your application in the green environment and test it thoroughly to ensure that it is working as expected.
3. Once the new version of your application is validated, switch the production traffic from the existing "blue" environment to the green environment.
4. If the new version experiences any issues, you can quickly switch the production traffic back to the blue environment.
5. After verifying that the new version is stable, you can decommission the blue environment.

AWS provides a range of tools and services that can help you to implement a blue-green deployment, such as Amazon ELB for load balancing, Amazon Route 53 for routing traffic, and AWS CloudFormation for automating deployment and rollback processes. By using these services, you can build a robust and reliable blue-green deployment process that ensures the stability and performance of applications.





In conclusion, blue-green deployment is a powerful deployment technique that enables you to roll out changes to your application with minimal downtime and risk. By using AWS services, you can build a robust and reliable blue-green deployment process that ensures the stability and performance of your applications and reduces the risk of introducing new features or updates that may impact their performance.

The advantages of blue-green deployment in AWS include:

1. **Minimal Downtime:** Blue-green deployment minimizes downtime during the deployment process, as traffic is only redirected from one environment to another once the new version of the application has been thoroughly tested and validated.
2. **Improved Testability:** Blue-green deployment allows you to test and validate changes in a production-like environment, reducing the risk of introducing bugs or performance issues into your application.
3. **Increased Availability:** Blue-green deployment ensures that your application is available and responsive even during a deployment, as traffic is redirected to the backup environment in case of any issues with the new version.
4. **Easier Rollback:** If the new version of your application experiences any issues, you can quickly roll back to the previous version by redirecting traffic back to the original environment.
5. **Improved Scalability:** Blue-green deployment allows you to scale your application in a controlled and measurable manner, ensuring that the new version is able to handle the traffic it receives.
6. **Better Resource Utilization:** Blue-green deployment allows you to use resources more effectively, as you can decommission the original environment once the new version has been deployed and validated.

For our deployments and releases we need to use CodeDeploy [13] service from AWS. This service makes it easier to automate deployment workflows, enforce consistency, and achieve seamless application updates.

It serves as an agile and scalable deployment solution that streamlines the deployment process for diverse applications. It empowers you to define customized deployment configurations, precisely specify the target environment, and automate the entire deployment workflow, thereby simplifying the release of new features and updates.

One of the advantages of CodeDeploy is its support for multiple deployment strategies, encompassing rolling deployments, blue/green deployments, and canary deployments. This remarkable flexibility enables you to select the most suitable strategy based on your application's unique requirements, while simultaneously minimizing downtime during the deployment process.

Furthermore, it offers robust rollback capabilities in the event of deployment failures or undesired impacts on your application's performance. You can quickly roll back to an earlier version, minimizing disturbance and guaranteeing your application's availability

and reliability even in complex deployment settings. This service provides comprehensive monitoring and visibility features, empowering you to closely track the progress of deployments, monitor the health of instances, and gain valuable insights into the deployment process through detailed logs. This high level of visibility allows you to proactively troubleshoot issues, identify performance bottlenecks, and continuously enhance your deployment workflows.

#### **4. Visibility and Traceability. AWS X-Ray**

Visibility and traceability are crucial aspects of cloud computing and are essential for ensuring the performance and security of applications running on the AWS platform. AWS provides several tools and services to help organizations achieve visibility and traceability in their cloud environments. One of the key tools for visibility and traceability in AWS is CloudTrail. It provides a detailed log of all AWS API calls, including the identity of the caller, the time of the call, and the response. This data can be used to monitor activity within the AWS environment, detect security threats, and perform auditing and compliance tasks. Another important tool for visibility and traceability in AWS is Amazon CloudWatch [14]. It provides a centralized view of the performance and health of an application, including the ability to monitor and visualize metrics, logs, and events.

AWS X-Ray is another tool that can help with visibility and traceability, it is a powerful tool for analyzing, debugging and visualizing the performance of distributed applications. It provides a detailed view of the requests and responses flowing through an application, making it easier to identify and resolve issues. One of the key benefits of using X-Ray is that it can be integrated with other AWS services, such as Amazon S3, Amazon DynamoDB, and Amazon Elasticsearch, to provide a complete view of the application. This allows developers to see how their application interacts with other services and how changes to those services may impact performance. X-Ray also provides a rich set of features for visualizing the performance of an application. The X-Ray service map shows a graphical representation of the request flow, making it easy to see which parts of the application are taking the most time. This can be helpful in identifying bottlenecks and performance issues. In addition to performance analysis, X-Ray also provides a tracing feature that allows developers to see the flow of requests through their application in real time. This can be especially useful for troubleshooting issues with production applications.

Another important feature of X-Ray is its integration with AWS Lambda. With X-Ray, developers can easily analyze the performance of their serverless functions, including the time spent in different parts of the application and the number of invocations.

Overall, AWS X-Ray is a valuable tool for any developer working with distributed applications on the AWS platform. Its ability to provide a comprehensive view of an application's performance, along with its integration with other AWS services, makes it a must-have for anyone looking to improve the performance and reliability of their applications.



In addition to performance analysis and troubleshooting, X-Ray also provides security features to help ensure the security of applications running on the AWS platform. X-Ray integrates with AWS WAF, a web application firewall, to provide protection against common web attacks such as SQL injection, cross-site scripting, and others. Another security feature is the ability to encrypt X-Ray data in transit and at rest. This helps to ensure that sensitive information remains confidential and secure, even when it is transmitted over public networks.

X-Ray also provides the ability to customize sampling rules. Sampling is the process of selecting a subset of requests to analyze, and X-Ray provides the ability to configure sampling rules based on custom attributes such as the request URL or the user's IP address. This allows developers to focus their analysis on the requests that are most important to them, and to avoid analyzing requests that are not relevant to their performance goals.

Finally, X-Ray provides a rich API for accessing and analyzing performance data. This API can be used to extract performance data from X-Ray and integrate it with other tools and services. For example, developers can use the API to extract performance data from X-Ray and import it into a time-series database for further analysis and visualization (figure 4).

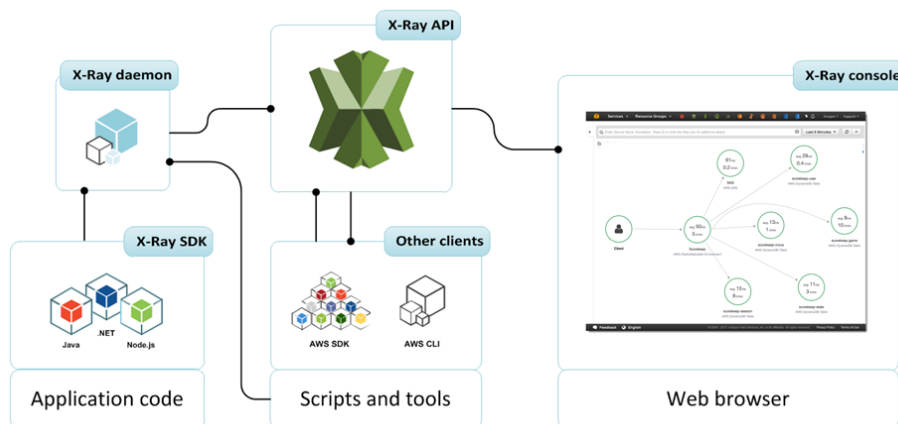


Figure 4 X-RAY [27]

For the X-RAY implementation we will need to use CloudWatch. It is a popular service provided by AWS that offers a wide range of monitoring, logging, and analysis capabilities to gain insights into the performance, health, and resource utilization of your applications and infrastructure.

As the central hub for monitoring and managing your AWS environment, CloudWatch enables you to effortlessly collect, track, and analyze metrics, monitor log files, set up alarms, and even automate responses to changes in your AWS resources. With CloudWatch, you gain real-time visibility into the operational health and performance of your applications and infrastructure.

One of the standout features of CloudWatch is its ability to collect and store metric data from various AWS services and custom sources. These metrics encompass critical performance indicators like CPU usage, network traffic, disk I/O, and an array of other valuable insights. Leveraging these metrics, you can discern patterns, identify trends, and troubleshoot issues within your environment with precision.

Moreover, CloudWatch Logs empowers you to effortlessly collect, monitor, and analyze log files originating from your applications and systems. By centralizing log data, leveraging the power of CloudWatch Logs Insights to search and filter logs, and setting up alarms based on log events, you gain an invaluable edge in identifying and resolving issues, tracking application behavior, and complying with security and auditing requirements. It offers a powerful feature that allows you to create customized dashboards, delivering a unified view of the health and performance of your AWS resources. These interactive dashboards provide real-time data and metrics tailored to your specific use cases, simplifying the monitoring and comprehension of your applications and infrastructure.

CloudWatch seamlessly integrates with other AWS services, empowering you to harness its data and insights for automated actions or advanced analytics. For example, CloudWatch Events can trigger AWS Lambda functions, enabling you to automate actions through the robust capabilities of AWS Systems Manager.

## **5. Securing. Amazon Cognito**

Amazon Cognito [15] is a cloud-based service offered by Amazon Web Services (AWS) that provides authentication and authorization for web and mobile applications. It helps developers build secure applications by providing user sign-up and sign-in functionality, as well as access control for their resources. Cognito enables developers to focus on building their applications, rather than writing custom authentication and authorization code. One of the key benefits of Amazon Cognito is its scalability. Cognito can handle millions of users, and the service automatically adjusts its capacity to meet the demands of your application. This eliminates the need for developers to worry about the underlying infrastructure required to support their authentication and authorization needs.

Another benefit of Amazon Cognito is its ease of use. The service integrates with other AWS services, such as AWS Lambda and Amazon S3, making it easy for developers to add authentication and authorization to their applications. Cognito also supports the SAML and OIDC protocols, enabling integration with existing identity providers such as Okta, Auth0, and Microsoft ADFS.

Cognito also provides a robust set of security features, including encryption of data at rest and in transit, and support for multi-factor authentication. The service is SOC, ISO, and PCI DSS compliant, making it a secure choice for storing and managing user identities.



To sum-up, Amazon Cognito is a powerful and flexible solution for adding authentication and authorization to your web and mobile applications. The service provides scalability, ease of use, and a robust set of security features, making it a great choice for developers looking to build secure and scalable applications. Whether you are building a new application or adding authentication and authorization to an existing one, Amazon Cognito can help you get there faster and with less hassle.

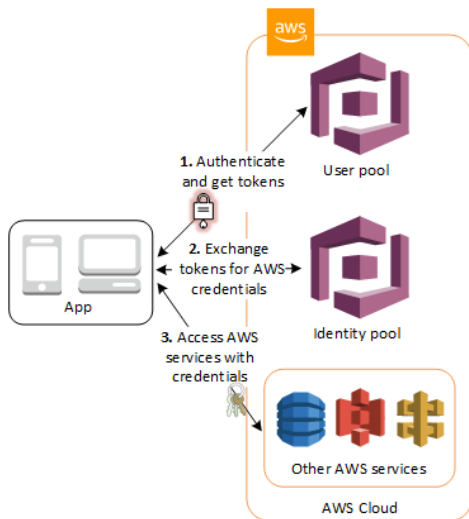


Figure 5 Amazon Cognito [28]

Cognito contains two main components:

- **User pools** - a user directory in Cognito where you can store user profile information, such as name and email address. User pools make it easy for developers to add user sign-up and sign-in functionality to their applications, without having to worry about the underlying infrastructure and security. One of the key advantages of using user pools in Cognito is that they provide a scalable solution for managing user identities. User pools can support millions of users, and the service automatically adjusts its capacity to meet the demands of your application. This eliminates the need for developers to worry about the infrastructure required to support their authentication and authorization needs. User pools in Cognito also provide a user-friendly sign-up and sign-in experience for your users. Cognito integrates with other AWS services, such as AWS Lambda and Amazon S3, making it easy for developers to add authentication and authorization to their applications. User pools also support the SAML and OIDC protocols, enabling integration with existing identity providers such as Okta, Auth0, and Microsoft ADFS. Another advantage of user pools in Cognito is the security features it provides. Cognito uses encryption to protect user data at rest and in transit, and provides support for multi-factor authentication. The service is SOC, ISO, and PCI DSS compliant, making it a secure choice for storing and managing user identities. In conclusion, user pools in AWS Cognito provide a secure and scalable solution for managing user identities for your web and mobile applications. Whether you are building a new application or adding authentication and authorization to an existing

one, user pools in Cognito can help you get there faster and with less hassle. With its scalability, ease of use, and robust security features, user pools in Cognito are a great choice for developers looking to build secure and scalable applications. One of the cases of how Cognito works is shown in figure 5.

**Identity pools** serve as a dedicated repository of user identity information that is tailored to an application's specific requirements and enables users to access AWS resources. Utilizing identity pools in AWS Cognito offers distinct advantages, notably granting users access to AWS services without necessitating the creation of an AWS account. This streamlined approach facilitates users' initial engagement with the application, effectively reducing barriers to entry. Furthermore, identity pools provide a mechanism for users to assume an AWS Identity and Access Management (IAM) role, enabling them to access AWS resources on behalf of the application. Cognito's identity pools also offer support for user authentication through diverse identity providers, such as social media platforms and enterprise directories. This simplifies the sign-up and sign-in processes for users, enhancing convenience and alleviating the burden on the application's management of user identities.

Another significant benefit of utilizing identity pools in AWS Cognito is the inherent scalability they provide. Designed to accommodate millions of users, identity pools seamlessly adjust their capacity to meet the demands of the application. Consequently, concerns regarding the infrastructure required to support user identities are effectively mitigated.

In summary, identity pools in AWS Cognito present a secure and scalable solution for facilitating users' access to AWS services. Whether embarking on the development of a new application or enhancing an existing one with robust authentication and authorization capabilities, leveraging identity pools in Cognito offers an expedited and streamlined approach. With their inherent scalability, user-friendly nature, and robust security features, identity pools in Cognito emerge as an excellent choice for developers aiming to construct secure and scalable applications.

## 6. Tools

For this project we need an AWS account. Amazon gives a free tier account for 12 months, where Tier allows new customers to try out AWS services and become familiar with them before committing to a paid account, as we can see on the Figure 6.



**AWS Free Tier**

Gain free, hands-on experience with the AWS platform, products, and services

[Learn more about AWS Free Tier](#)

[Create a Free Account](#)

**FEATURED**  
**Startups may be eligible for AWS credits**  
AWS Activate provides eligible startups with a host of resources, including free AWS credits to spend on AWS services, and AWS Support.  
[Sign up for Activate Today »](#)

### Types Of Offers

Explore more than 100 products and start building on AWS using the Free Tier. Three different types of free offers are available depending on the product used. Click icon below to explore our offers.

- Free trials**  
Short-term free trial offers start from the date you activate a particular service
- 12 months free**  
Enjoy these offers for 12-months following your initial sign-up date to AWS
- Always free**  
These free tier offers do not expire and are available to all AWS customers

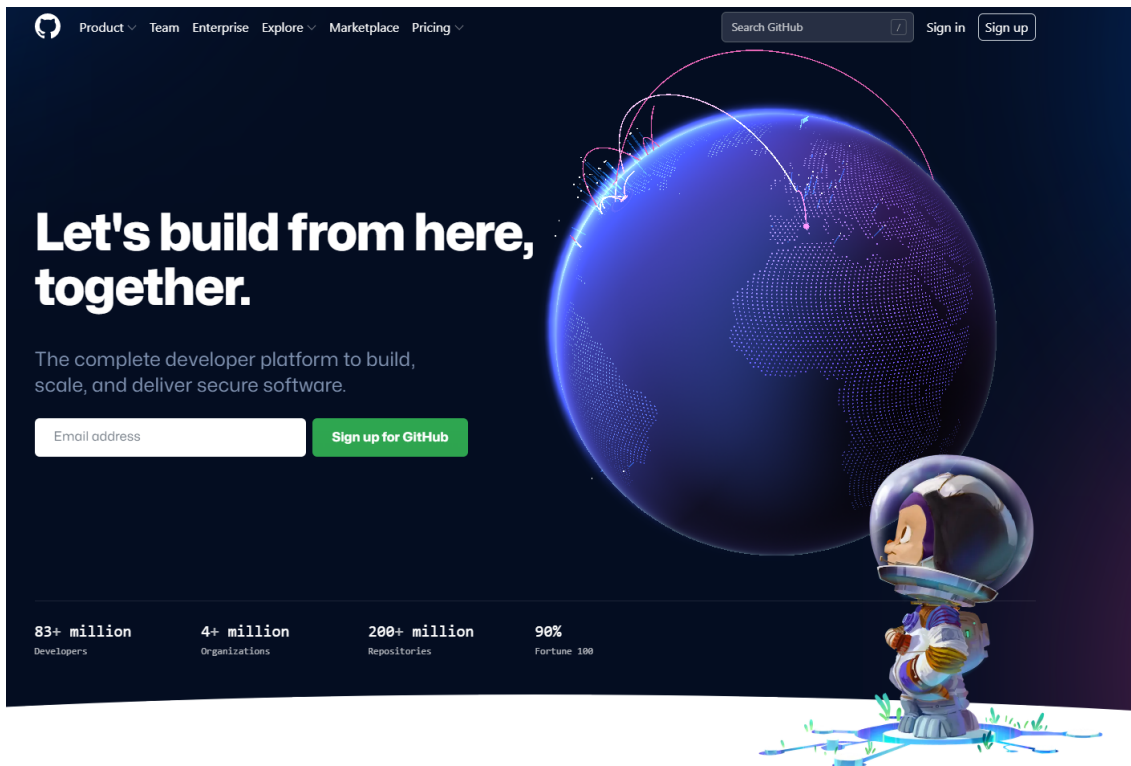
Figure 6 Free tier account

The AWS Free Tier offers users access to a diverse selection of services, encompassing computing, storage, databases, and analytics. For instance, customers can leverage a t2.micro instance in Amazon Elastic Compute Cloud (EC2) for up to 750 hours per month at no cost. They can also utilize Amazon Simple Storage Service (S3) to store data up to 5 GB and make use of Amazon DynamoDB for up to 25 GB of data storage without incurring charges.

To take advantage of the AWS Free Tier, users simply need to create an AWS account. Once the account is set up, they can immediately begin utilizing the available free services. If the usage surpasses the limits defined by the free tier, any additional usage will be subject to charges. However, customers are provided with a 12-month grace period to transition to a paid account if they require more services than what the free tier offers.

The AWS Free Tier serves as an excellent starting point for new customers to explore and gain hands-on experience with AWS services. Whether they are building new applications, testing existing ones, or simply exploring the capabilities of AWS, the free tier provides a valuable resource to get started. With its extensive array of services, generous usage limits, and flexible pricing options, the AWS Free Tier offers an accessible opportunity to familiarize oneself with AWS and its benefits.

Additionally, it is necessary to have a git account on any platform like GitHub, GitLab, BitBucket, etc. In this case, GitHub has been selected due to its popularity, prior experience, stability, student-friendly features, and availability at no cost. This account is going to be used for forking an example application into our repository for future work.



*Figure 7 GitHub page*

GitHub is a web-based platform that offers software development hosting and version control using Git. It was established in 2008 and later acquired by Microsoft in 2018. As the largest host of source code globally, GitHub boasts a user base exceeding 40 million and hosts over 100 million repositories. By creating branches within a repository, users can work on separate features and subsequently merge their changes back into the main branch. Additionally, pull requests allow for proposed code modifications, which can be reviewed and approved by other team members.

Another noteworthy aspect of GitHub is its extensive collection of open-source projects, enabling developers to easily utilize and customize existing codebases. This streamlines the process of building new applications, fostering efficiency and agility. Furthermore, GitHub provides an array of project management tools such as issues, milestones, and project boards, aiding teams in organizing and tracking their progress effectively.

In addition to its collaborative nature, GitHub incorporates tools for ensuring code quality and security. GitHub Actions enables the automation of software development workflows, while GitHub Security Alerts notifies users of potential security vulnerabilities in their code.

To sum up, GitHub stands as a robust platform for software development and collaboration, widely embraced by millions of developers worldwide. Its collaborative features, vast repository of open-source projects, and tools for code quality and security make it an indispensable tool for software development teams aiming to enhance productivity and efficiency.





The next important thing is IDE. Nowadays, there are a lot of development environments, and it is not a big problem to find such a good product depending on different needs. For this project, Visual Studio Code has been selected because it is completely free and meets all the requirements.

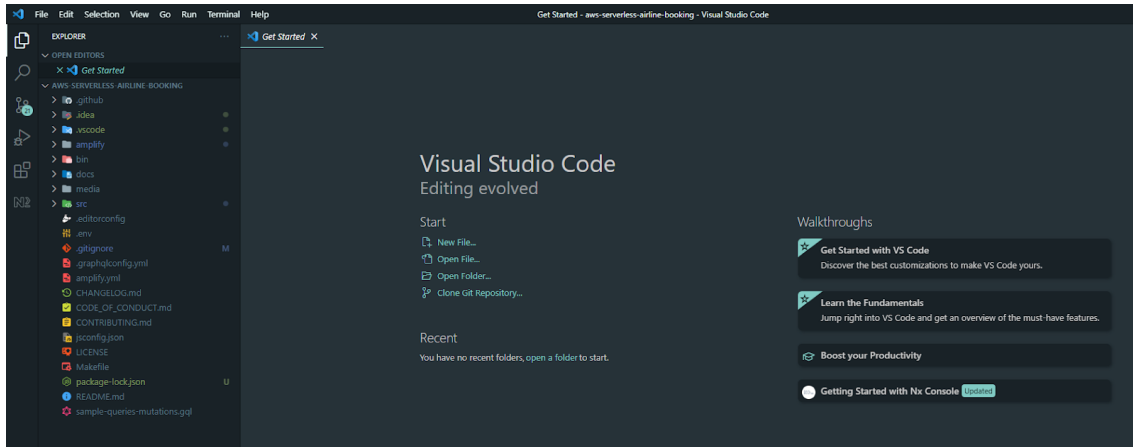


Figure 8 Visual studio code

Visual Studio Code (VSCode) is a free and open-source code editor, created by Microsoft. One of the main advantages of VSCode is cross-platform compatibility, supporting Windows, macOS and Linux.

VSCode has a lot of plugins and extensions, and it is also updating with new features and functionalities, such as debugging, code completion, and code analysis. This customizable aspect gives a possibility to configure VSCode to specific requirements. With its extensive functionality and wide-ranging features, Visual Studio Code offers a compelling solution for developers. Its user interface is not only versatile but also user-friendly, making it accessible even to beginners in coding (figure 8).

All the front-end part is using JavaScript as the main language. **JavaScript** is a highly popular high-level programming language [16], empowers web developers to bring interactive and dynamic functionalities to websites. Its versatility and expressive syntax provide a seamless experience for crafting responsive user interfaces, manipulating data, and interacting with web APIs. Supported by contemporary web browsers, JavaScript stands as a foundational language in the realm of web development. Its adaptability and vast library ecosystem extend its utility to both front-end and back-end scenarios.

For the application, it is necessary to install:

- Python
- Node.js
- Vue.js
- Quasar framework
- SAM CLI
- Docker

Nowadays, **Python** is one of the most popular, high-level programming languages known for its simplicity, readability, and versatility. It has clean, concise syntax and promotes code readability. Apart from the standard library and a vast ecosystem of third-party modules and frameworks, Python enables developers to build a wide range of applications, from web development and scientific computing to data analysis and artificial intelligence. Additionally, Python's cross-platform compatibility ensures that applications written in Python can run seamlessly on various operating systems.

JavaScript code may now be executed outside a web browser thanks to the robust, adaptable and open-source runtime environment **Node.js**. It is the perfect option for developing scalable and high-performance server-side applications because of its event-driven and non-blocking I/O style, which ensures effective handling of numerous concurrent requests. Developers have access to a wide number of tools and functions through its robust ecosystem of modules and libraries, enabling them to produce original and ground-breaking solutions.

**Vue.js** is a lightweight and powerful JavaScript framework designed for building dynamic and interactive user interfaces. It offers a modular and component-based architecture, making it easy to develop and maintain complex UIs. With its reactivity system and efficient rendering, this framework enables seamless data binding and real-time updates. It boasts a thriving ecosystem and strong community support, solidifying its position as a popular choice for modern web development.

The **Quasar Framework** is an open-source framework that expands the capabilities of Vue.js. This framework is quite popular among developers to construct web and mobile applications that are both highly efficient and responsive, all while utilizing a single codebase. By offering an extensive array of pre-built components, themes, and plugins, Quasar simplifies the development process and facilitates swift prototyping. Furthermore, its adaptable design system guarantees seamless compatibility across a wide range of platforms and screen sizes, resulting in a consistent and optimal user experience. Through its wide range of customization options, developers can personalize their applications to meet specific requirements and construct interfaces that are truly distinct. The Quasar Framework is efficient, scalable, and cross-platform, making this tool so popular for creating cutting-edge applications fast and simple.

**AWS SAM CLI**, or the AWS Serverless Application Model Command Line Interface, is a tool that allows developers to manage their serverless applications on the Amazon Web Services (AWS) cloud platform [17]. It provides a local development environment for building, testing, and debugging serverless applications and functions built using the AWS Serverless Application Model (SAM) framework.

One of the key advantages of the AWS SAM CLI is its ease of use. It provides a simple, intuitive command line interface that makes it easy to manage serverless applications and functions. This helps to streamline the development process and reduce the time and effort required to build, test, and deploy serverless applications.

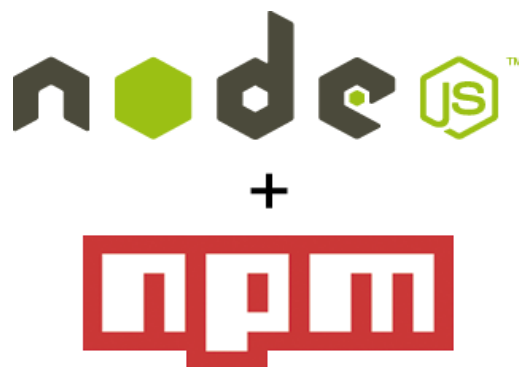


Another advantage of the AWS SAM CLI is its integration with AWS services. The CLI integrates seamlessly with other AWS services, such as AWS Lambda, Amazon S3, and Amazon API Gateway, making it easy to manage serverless applications and functions built using these services. This integration allows developers to quickly and easily manage their serverless applications, without having to use multiple tools or navigate complex AWS console interfaces.

Another noteworthy strength of the AWS SAM CLI is its integration with various AWS services. Integrating seamlessly with services like AWS Lambda, Amazon S3, and Amazon API Gateway, the CLI facilitates the management of serverless applications and functions developed using these services.

**Docker** is an open-source platform that provides containerization technology for developing, deploying, and running applications. It allows developers to package their applications along with their dependencies into lightweight and portable containers. These containers can then be easily deployed on any system that has Docker installed, ensuring consistent and reliable application execution across different environments. Docker simplifies the software development and deployment process by abstracting away the underlying infrastructure, enabling efficient resource utilization and scalability.

Moreover, when working with JavaScript modules, it is recommended to utilize a package manager called NPM. NPM offers a lot of advantages, including its free and user-friendly, extensive documentation, and the ability to swiftly install all the required libraries with a single command. Leveraging the power of NPM streamlines the management of JavaScript modules, enabling developers to easily integrate and utilize the necessary libraries in their projects.



*Figure 9 NodeJs & npm [29]*

**NPM (Node Package Manager)**, a package manager for the JavaScript programming language, offers a comprehensive solution for managing and distributing code packages, including libraries and tools, specifically designed for Node.js applications. NPM streamlines the process of integrating, updating, and sharing packages, significantly simplifying the development and maintenance of applications.

# Chapter 3. Exemplary Microservices-based Application

---

This chapter describes the use of all the technologies and tools that are needed for the application and its architecture.

## 1. Application architecture

The application that is used for this project is open source from Amazon Web Services and could be found on their GitHub account “aws-samples”. It was created for the testing needs and has the next structure (figure 10):

### 1) Front-end

The front-end of the application is built using Vue.js as the core framework, providing a robust and scalable foundation. Quasar is utilized for the UI, offering a comprehensive set of components and tools for creating a visually appealing user interface. Amplify is incorporated to streamline the authentication process and enable seamless integration with AWS services, ensuring a secure and efficient user experience.

### 2) Data

All data within the application is structured according to GraphQL types, promoting efficient and flexible data management. DynamoDB serves as the primary data storage solution, leveraging its scalability and high-performance capabilities. Python is the core language utilized for all services. JavaScript is employed for the front-end development, ensuring a cohesive and unified coding environment.

### 3) API

The application's API layer is managed by AppSync, which acts as a centralized hub for GraphQL interactions with other services. AppSync provides a seamless integration between different components, facilitating efficient data retrieval and manipulation.

### 4) Auth

Cognito, a robust authentication and user management service, is utilized to provide JSON Web Tokens (JWT) for secure authentication. Together with AppSync, it enables fine-grained authorization control, allowing administrators



to define access privileges for different user roles. This ensures that users can only access the data types and resources that are authorized for their specific roles.

## 5) Messaging

Step Functions handle the process for bookings, while SNS handles service-to-service messaging between Booking and Loyalty.

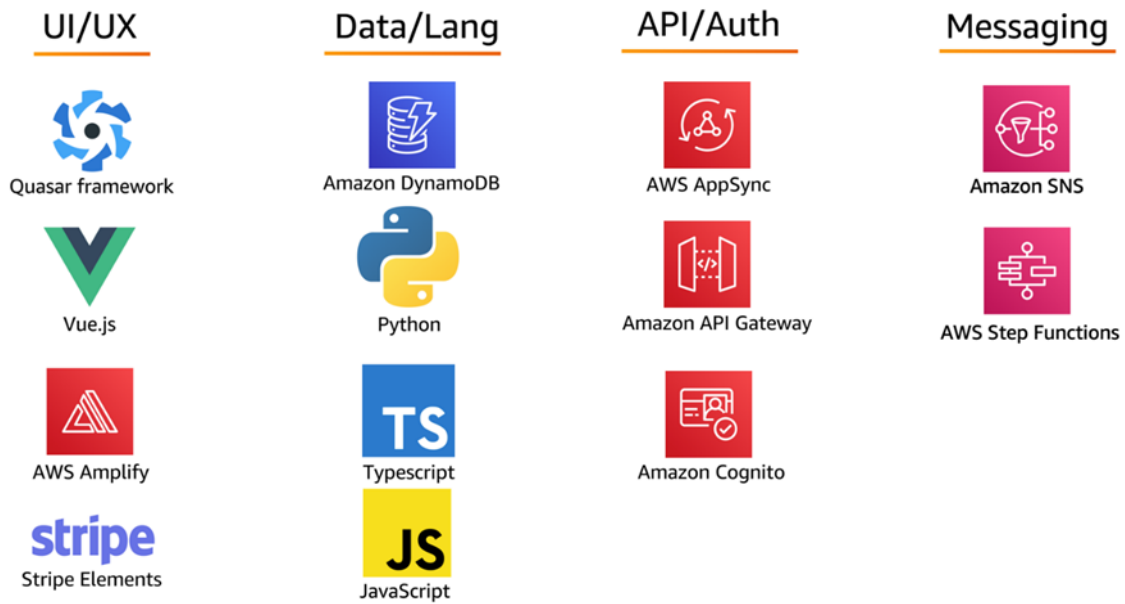


Figure 10 NodeJs & npm [30]

## 2. Project structure

As it was mentioned before, this project will analyze different strategies, tools and services such as AWS X-RAY, Amplify, Lambda functions, etc. It has five parts: Deployment, Scaling, Updating, Visibility and Securing. As shown in Figure 11, it has the following structure:

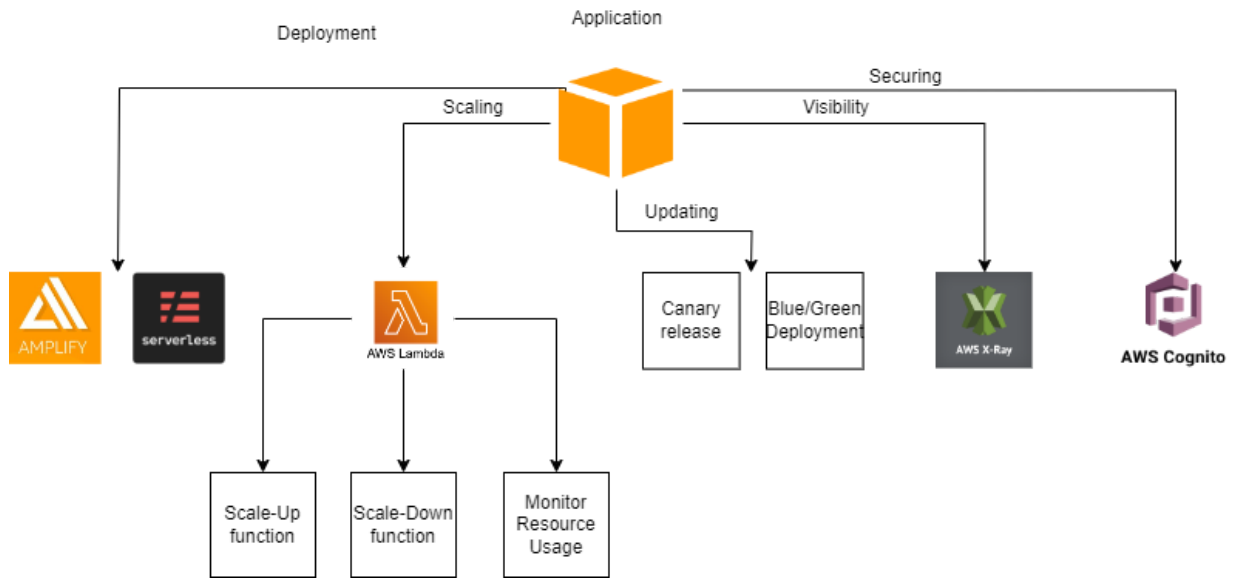


Figure 11 Project structure

The first step is to deploy the application example in AWS. Thanks to Amplify and serverless framework this process will not take too much time and will do most of the deployment automatically. The deployment part will be discussed in detail in chapter 4 of this project.

The next step is to configure the authentication service with Cognito. It sets up the registration, authentication, creates users, identity pools and setup required security layers. Thanks to this service it is possible to configure the sign in and sign up processes with the specific requirements and needs.

After that, it is essential to create lambda functions for different scenarios for the scaling part of the application as shown in figure 12.



# Life Cycle Management of Serverless Microservices using Amazon Web Services

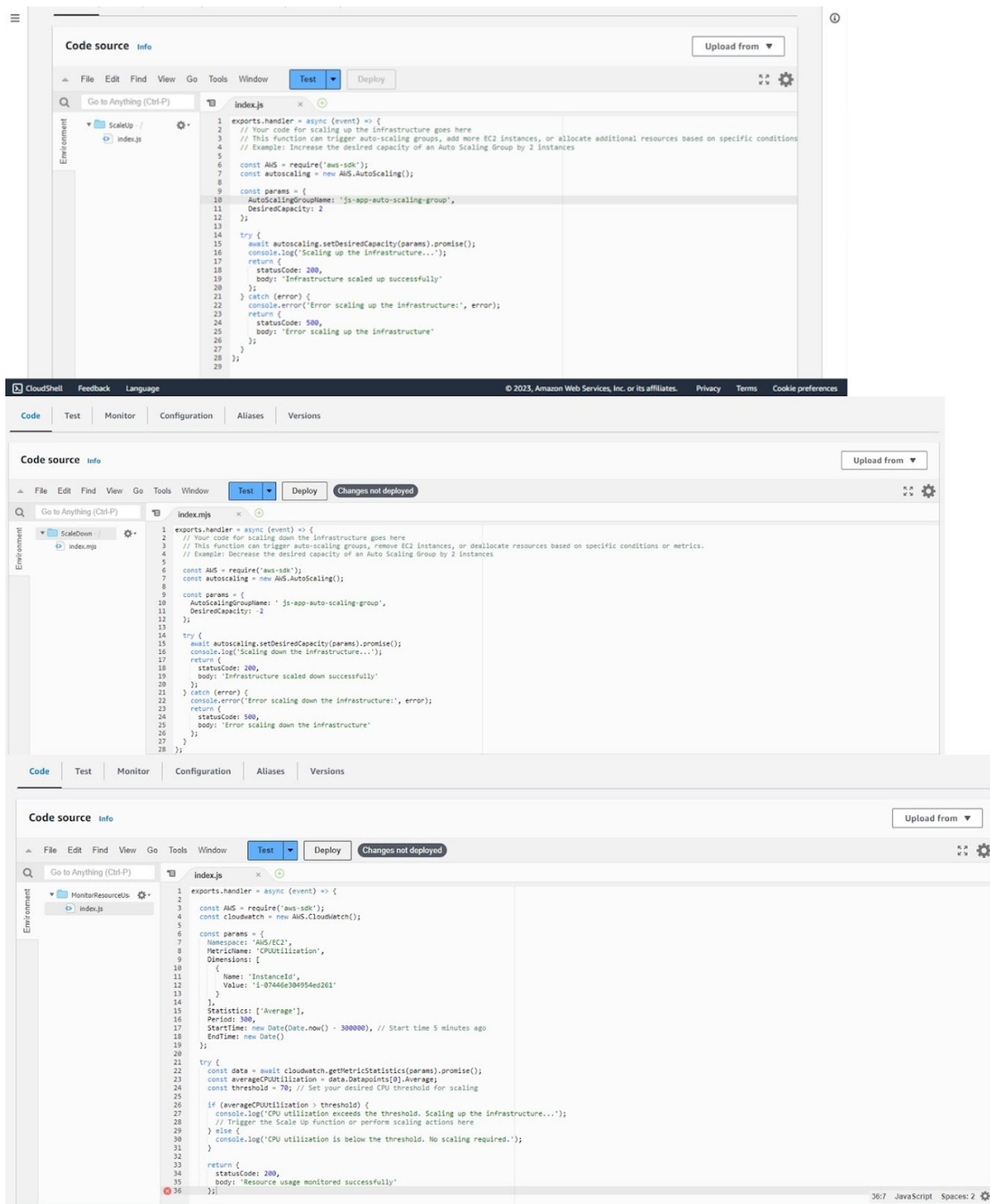


Figure 12 Lambda functions

For the update part, Canary Release and Blue/Green deployment would be implemented using CodeDeploy, CloudWatch, IAM and other AWS services.

The last step is to use the X-RAY SDK in the project and then connect AWS X-RAY to analyze and find errors in the application. All the required steps and code are described in the next chapter.

# Chapter 4. Deployment

---

This section describes the whole process of configuration of the application in AWS.

## 1. Deploy

We deploy our frontend section of the project with Amplify Hosting; it will do it automatically [18]. AWS (Amazon Web Services) offers a hosting platform called Amplify Hosting that makes it easy and affordable to host static websites and web apps. It offers a range of features that make it simple to create and scale applications, such as content delivery, secure authentication, and continuous deployment.

The simplicity of usage of Amplify Hosting is one of its main benefits. Without having to worry about underlying infrastructure or scalability difficulties, Amplify enables you to rapidly design and launch a web application or static website. So that you can concentrate on creating and enhancing your application, it automatically handles the deployment and administration of your application.

The affordability of Amplify Hosting is another benefit. Because of Amplify's flexible pricing structure, you only pay for the resources you really utilize. Because they do not have the funds for more expensive hosting options, startups and small enterprises find it to be an appealing choice.

Moreover, Amplify Hosting offers a high level of security. Amplify protects your application and its data using Amazon security features including encryption, access restrictions, and logging. This gives you peace of mind while assisting in the protection of your application and its users from security risks.

To offer a comprehensive solution for developing, deploying, and maintaining web apps, Amplify also connects with other Amazon services like Lambda and AppSync. This connection offers developers a smooth experience while making it simple to add features and functionality to your application.

Last but not least, Amplify Hosting is a reliable and cost-effective hosting service offered by Amazon. Because of its ease of use, affordability, security, and integration with other Amazon services, it is the best option for developers and businesses wishing to create and extend static websites and online apps. Whether you are a newbie or skilled developer, Amplify Hosting is assured to satisfy your requirements and help you achieve your goals.

To get started, it is necessary to link a personal GitHub account, after which it will create a fork of this project under this account, the next step is to enter the name of the application and select an Amplify role.





# Life Cycle Management of Serverless Microservices using Amazon Web Services



Figure 13 Amplify hosting

If there are no roles created yet, it could be created as it is shown on Figure 14.

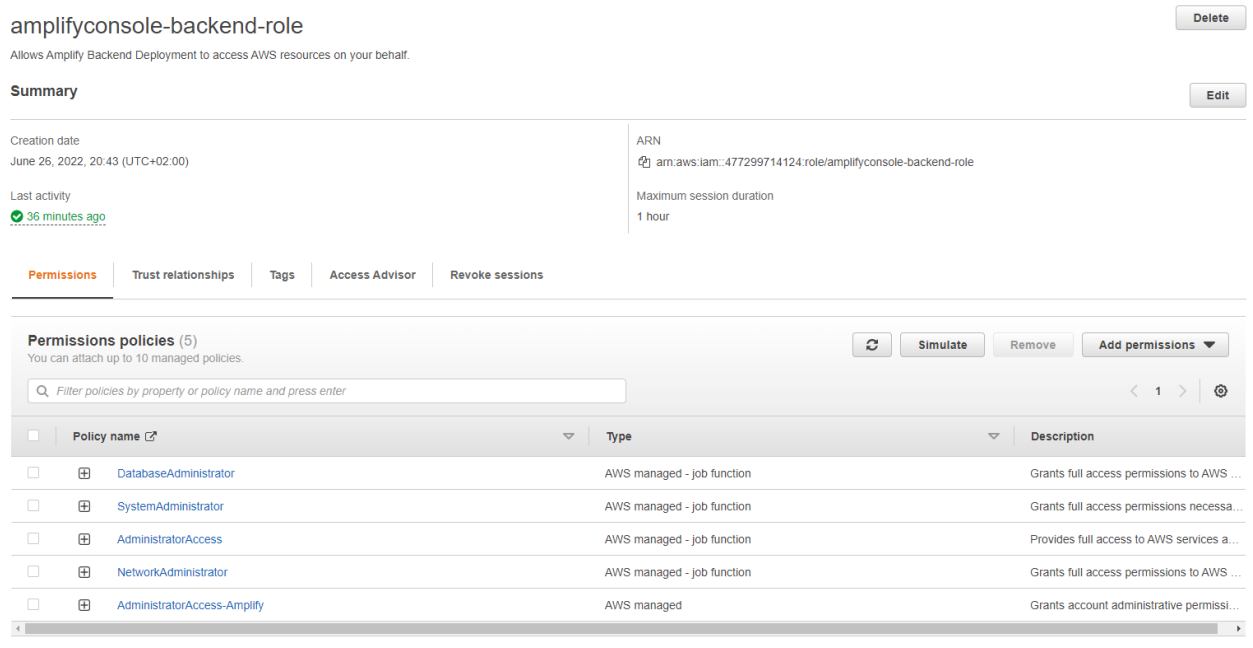


Figure 14 Permissions policies

On the last step of deploying, it shows a page of executed actions and the link on the deployed application (Figures 15 and 16).

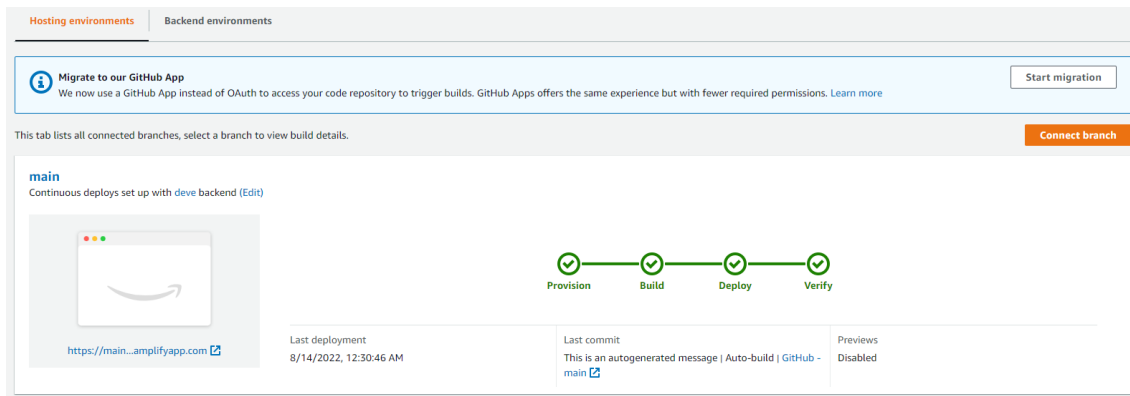


Figure 15 Deployed application

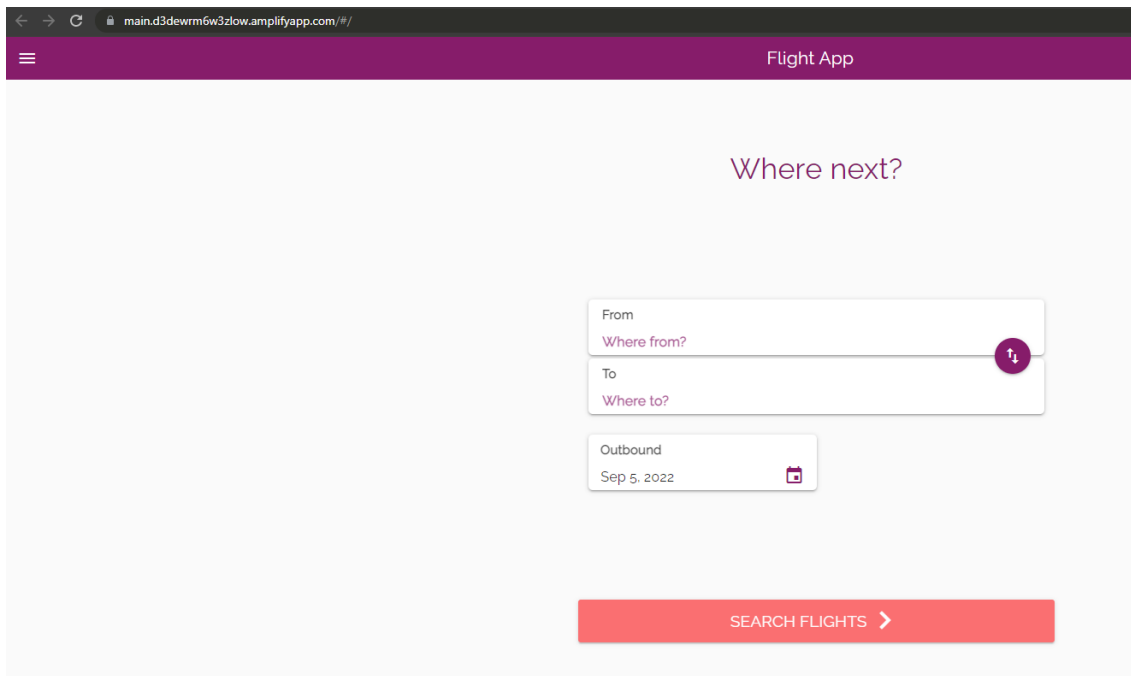


Figure 16 Deployed application on AWS

The next step is to deploy the backend part, so in the terminal, where the Amplify CLI is already installed, it is necessary to type ***amplify configure***. Thanks to this command, all the required data for the amplify account can be indicated and it makes a deployment via terminal:

- 1) Sign into your personal AWS administrator account → it will open an authentication form in the browser.
- 2) The specific region where the application deployed has been indicated. The Central Europe zone - Frankfurt with code “*eu-central-1*” has been chosen.
- 3) Username → After the name has been placed, it opens an IAM form to create a user with permissions and policies that are needed to deploy.
- 4) Introduce an Access key ID of the user.
- 5) Introduce a Secret access key of the user.
- 6) Profile Name: This would update/create the AWS Profile in the local machine.

As a result, a new user in the IAM service was created (figure 17), setted it locally and configured Amplify on the local machine (figure 18).

## Add user

1 2 3 4 5

✔ **Success**

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://477299714124.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key
▶	✔ amplify-test	AKIAW6IKHVBGB2C77YND <span style="font-size: small;">🔗</span>	***** <span style="font-size: small;">Show</span>

Figure 17 IAM user

```
D:\Alex\TFG\aws-serverless-airline-booking>amplify configure
Follow these steps to set up access to your AWS account:

Sign in to your AWS administrator account:
https://console.aws.amazon.com/
Press Enter to continue

Specify the AWS Region
? region: eu-central-1
Specify the username of the new IAM user:
? user name: amplify-test
Complete the user creation using the AWS console
https://console.aws.amazon.com/iam/home?region=eu-central-1#/users$new?step=final&accessKey&userNames=amplify-test&permissionType=policies&policies=arn:aws:iam::aws:policy%2FAdministratorAccess-Amplify
Press Enter to continue

Enter the access key of the newly created user:
? accessKeyId: *****
? secretAccessKey: *****
This would update/create the AWS Profile in your local machine
? Profile Name: default

Successfully set up the new user.
```

Figure 18 Amplify initialization

The next step is to initialize a project with Amplify command ***amplify init***:

- 1) Enter a name for the environment.
- 2) Select the authentication method → it has options to choose between AWS access keys, AWS profile and Amplify Studio. In this case AWS Profile has been selected.
- 3) Indicate a profile to use (this profile has been created in the step before).

It shows a message that the project has been successfully initialized and connected to the cloud (Figure 19).

```

D:\Alex\TFG\aws-serverless-airline-booking>amplify init
Note: It is recommended to run this command from the root of your app directory
? Do you want to use an existing environment? No
? Enter a name for the environment test
Using default provider awscloudformation
? Select the authentication method you want to use: AWS profile

For more information on AWS Profiles, see:
https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html
? Please choose the profile you want to use default
Adding backend environment test to AWS Amplify app: d3dewrm6w3z1ow
? Initializing project in the cloud...

CREATE_IN_PROGRESS amplify-awsserverlessairline-test-223000 AWS::CloudFormation::Stack Mon Sep 05 2022 22:30:02 GMT+0200 (Central European Summer Time) User Initiated
CREATE_IN_PROGRESS UnauthRole AWS::IAM::Role Mon Sep 05 2022 22:30:08 GMT+0200 (Central European Summer Time) Resource creation Initiated
CREATE_IN_PROGRESS DeploymentBucket AWS::S3::Bucket Mon Sep 05 2022 22:30:09 GMT+0200 (Central European Summer Time) Resource creation Initiated
? Initializing project in the cloud...

CREATE_IN_PROGRESS AuthRole AWS::IAM::Role Mon Sep 05 2022 22:30:08 GMT+0200 (Central European Summer Time) Resource creation Initiated
? Initializing project in the cloud...

CREATE_IN_PROGRESS UnauthRole AWS::IAM::Role Mon Sep 05 2022 22:30:08 GMT+0200 (Central European Summer Time) Resource creation Initiated
CREATE_IN_PROGRESS DeploymentBucket AWS::S3::Bucket Mon Sep 05 2022 22:30:09 GMT+0200 (Central European Summer Time) Resource creation Initiated
CREATE_IN_PROGRESS AuthRole AWS::IAM::Role Mon Sep 05 2022 22:30:09 GMT+0200 (Central European Summer Time) Resource creation Initiated
? Initializing project in the cloud...

CREATE_COMPLETE UnauthRole AWS::IAM::Role Mon Sep 05 2022 22:30:26 GMT+0200 (Central European Summer Time)
CREATE_COMPLETE AuthRole AWS::IAM::Role Mon Sep 05 2022 22:30:27 GMT+0200 (Central European Summer Time)
? Initializing project in the cloud...

CREATE_COMPLETE DeploymentBucket AWS::S3::Bucket Mon Sep 05 2022 22:30:30 GMT+0200 (Central European Summer Time)
CREATE_COMPLETE amplify-awsserverlessairline-test-223000 AWS::CloudFormation::Stack Mon Sep 05 2022 22:30:32 GMT+0200 (Central European Summer Time)
? Successfully created initial AWS cloud resources for deployments.
? Initialized provider successfully.
? Initialized your environment successfully.
B Your project has been successfully initialized and connected to the cloud!

Some next steps:
"amplify status" will show you what you've added already and if it's locally configured or deployed
"amplify add <category>" will allow you to add features like user login or a backend API
"amplify push" will build all your local backend resources and provision it in the cloud
"amplify console" to open the Amplify Console and view your project status
"amplify publish" will build all your local backend and frontend resources (if you have hosting category added) and provision it in the cloud

Pro tip:
Try "amplify add api" to create a backend API and then "amplify push" to deploy everything

```

Figure 19 Amplify result

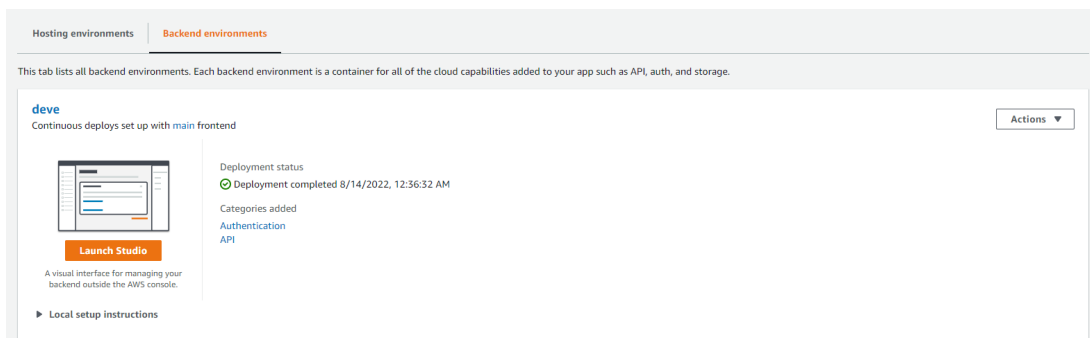


Figure 20 Backend environment

On Figure 20 the final part of the application’s deployment is shown.

## 2. Scaling. Lambda functions

As it was described before, Lambda functions give us a possibility to execute our code without any server. These functions provide automatic scalability, dynamically adjusting the provisioned infrastructure based on incoming request rates. This eliminates the need for manual capacity planning and allows applications to quickly respond to changing workloads. The functions scale horizontally by adding more instances to distribute the workload during high request volumes and scaling down during low periods to optimize resource utilization. AWS Lambda supports an event-driven architecture, seamlessly integrating with other AWS services to trigger functions based on specific events. Scaling behavior could be customized through configurable settings, ensuring optimal performance and resource allocation. With automatic scalability, Lambda functions enable applications to handle traffic spikes, accommodate varying workloads, and deliver a seamless user experience without manual intervention.



Provisioned Concurrency in AWS Lambda functions is a robust capability that guarantees consistent performance and minimized cold start durations. By leveraging provisioned concurrency, it is possible to pre-warm Lambda function instances to sustain a predetermined level of capacity, thereby enabling prompt response to incoming requests.

Through the configuration of provisioned concurrency, a fixed number of function instances are allocated and kept in a warmed state, readily available to process requests. This eradicates the necessity for cold starts, which transpire when a function instance is initiated from scratch to handle a request. The presence of pre-warmed instances significantly diminishes latency, ensuring anticipated response times for applications.

The advantages of provisioned concurrency are particularly pronounced in applications that necessitate stringent latency requirements or confront sudden surges in traffic. By ensuring the availability of warm instances at all times, it upholds responsiveness even in high-demand scenarios.

This feature is adjustable and can be tailored to suit the specific needs of an application by defining the desired number of provisioned concurrency instances. Furthermore, the allocation can be dynamically modified to scale up or down in response to changing traffic patterns.

Provisioned concurrency empowers developers to optimize the performance of Lambda functions and furnish users with a consistent and seamless experience. By eliminating cold starts and reducing latency, it facilitates swifter response times, ensuring that the application is always primed to handle incoming requests. For example, for the scenarios with 10,000 requests (figure 21), the function is configured with a Provisioned Concurrency of 7,000 [23]:

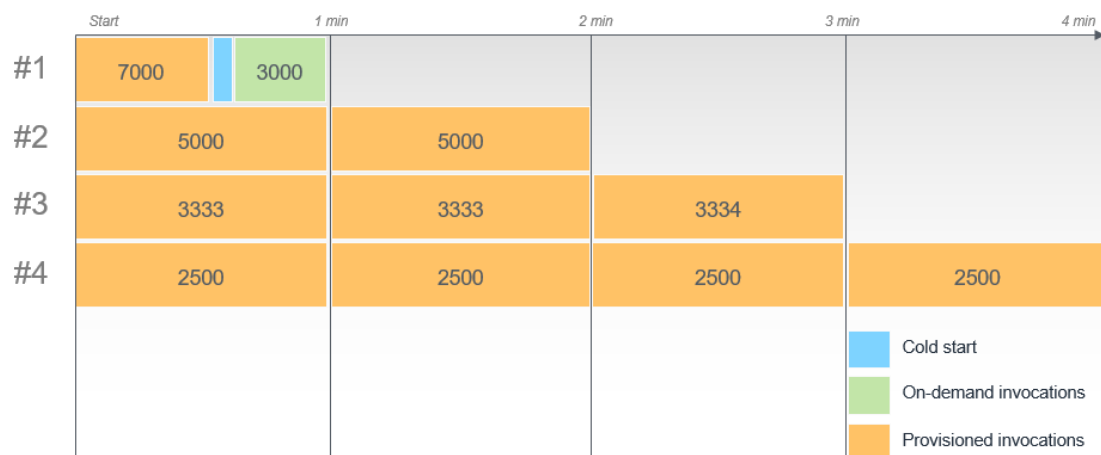


Figure 21 Provisioned Concurrency [31]

In case #1, 7,000 requests are handled by the provisioned environments with no cold start. The remaining 3,000 requests are handled by new, on-demand execution environments.

In cases #2-4, all requests are handled by provisioned environments in the minute when they arrive.

Serverless autoscaling of Lambda functions is a dynamic and efficient mechanism that automatically adjusts function capacity based on workload. With this capability, it is possible to ensure optimal resource allocation and responsiveness without manual scaling.

By leveraging serverless autoscaling, Lambda functions seamlessly adapt to changes in demand, scaling up or down to handle workloads. Additional instances are provisioned during high demand to process requests promptly. During low demand, the infrastructure scales down to optimize resource utilization and minimize costs.

This autoscaling capability is inherent in the serverless architecture. AWS Lambda, along with services like Amazon CloudWatch, monitors the workload and adjusts scaling automatically.

Scaling policies could be defined based on metrics such as request count, latency, or error rates. These policies align the number of instances with the workload, ensuring optimal performance and cost efficiency.

Serverless autoscaling is particularly advantageous for applications with unpredictable workloads. It enables scalability and elasticity, allowing applications to handle spikes in traffic and optimize resource utilization.

Quite often there are applications that have one part of the services serverless and another one that uses EC2 instances. That happens with the services whose requirements exceed the capabilities of Lambda. This kind of application is called “Hybrid application” on which the scaling part of this project was focused. Lambda functions were created for scale control of EC2 instances.

For the application scaling [19] it is essential to control some cases such as: scale-up/scale-down of traffic and monitor resource usage. For these scenarios three Lambda functions are implemented.

To create a new function, we must complete the next requirements:

- Enter a function name, it is better to put a name that best describes the functionality.
- Choose the language to use to write our function, it supports .NET/Java/Python/Ruby/Node.js/Go.
- Choose an architecture between x86\_64 and arm64. Arm64 – 64-bit ARM architecture, for the AWS Graviton2 processor and x86\_64 – 64-bit x86 architecture, for x86-based processors.
- Set an execution role. By default, it could create an execution role with basic permissions, and later it could be modified according to your needs. Another



option is to use an existing one (it could be created before using IAM Service with the required policies).

Also, it has Advanced Settings where we could:

- Enable Code signing
- Enable function URL
- Enable tags
- Enable VPC

After creating a function, it opens a Code editor where we could put our function code. Besides, we could create a JSON file that would be used as input data for our tests. In this case we are using JavaScript. After completing the code part, it could be tested without any server. We could put a trigger on how this event would be executed (API Gateway, AWS IoT, DynamoDB, CloudWatch, etc.), destination, it gives us a possibility to monitor the execution, how much memory was used, process time, logs, errors, etc.

### First Function: Scale-Up Task

The purpose of this function is to automatically scale up the infrastructure based on a predefined threshold, such as increased traffic or workload demand.

The function code:

```
exports.handler = async (event) => {  
  
  const AWS = require('aws-sdk');  
  const autoscaling = new AWS.AutoScaling();  
  
  const params = {  
    AutoScalingGroupName: 'js-app-auto-scaling-group',  
    DesiredCapacity: 2  
  };  
  
  try {  
    await autoscaling.setDesiredCapacity(params).promise();  
    console.log('Scaling up the infrastructure...');  
    return {  
      statusCode: 200,  
      body: 'Infrastructure scaled up successfully'  
    };  
  } catch (error) {  
    console.error('Error scaling up the infrastructure:',  
error);  
    return {  
      statusCode: 500,  
      body: 'Error scaling up the infrastructure'  
    };  
  }  
};
```

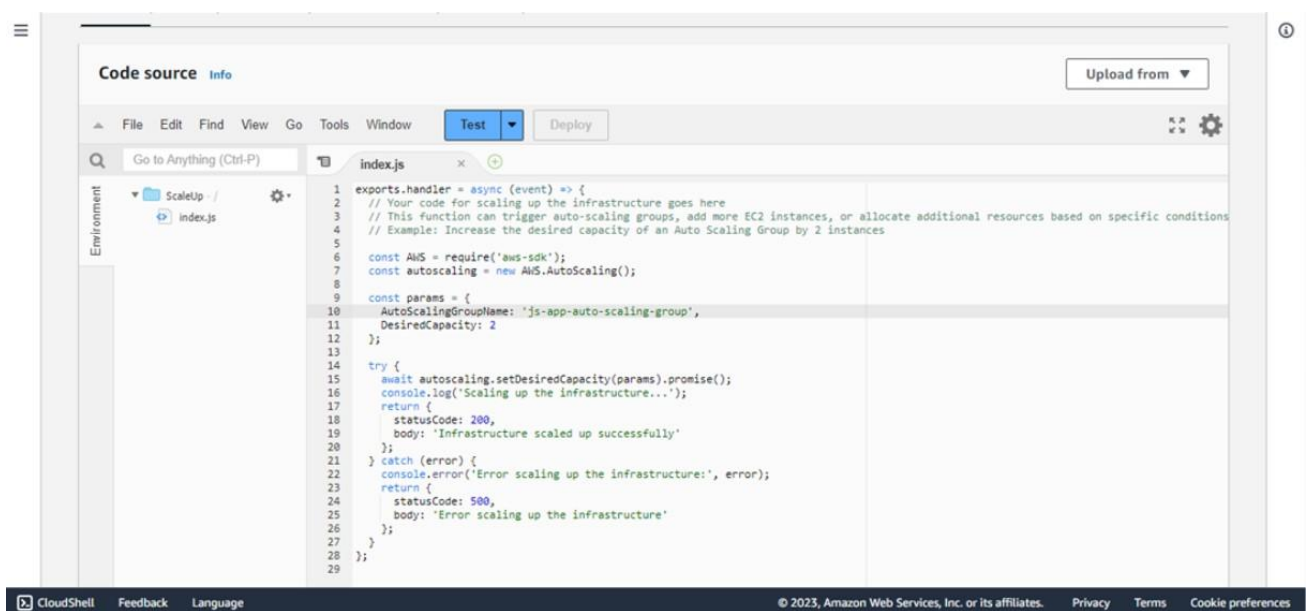
The purpose of the "Scale Up" feature is to enhance the capacity or resources of the infrastructure in response to increasing demand or workload. This functionality is

commonly employed in scenarios where automatic scaling is configured, such as an Auto Scaling Group within the AWS environment.

The "Scale Up" function operates by adjusting the desired capacity of the Auto Scaling Group, effectively introducing additional instances to the infrastructure. Utilizing the AWS SDK, this function interacts with the Auto Scaling service and sets the desired capacity to a higher value.

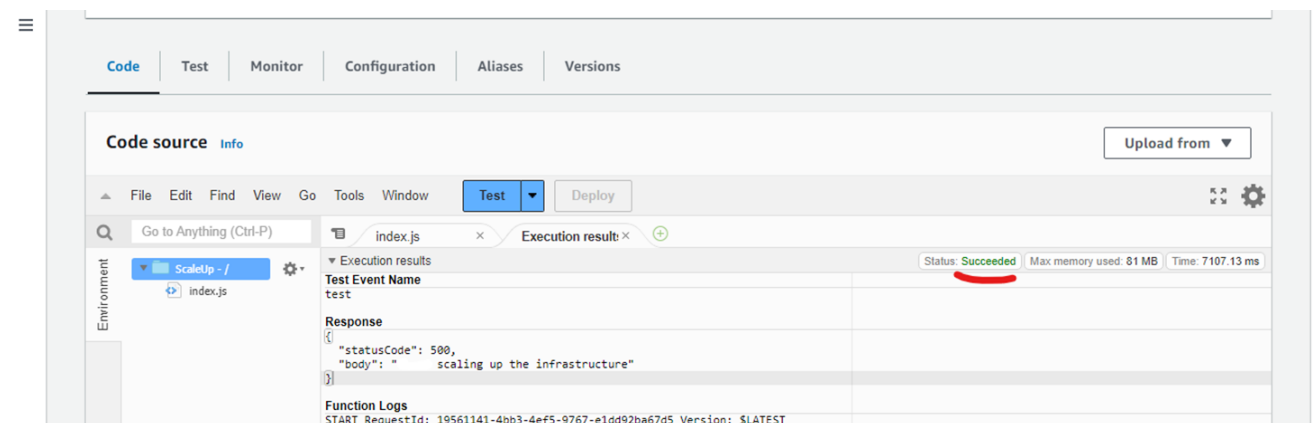
By invoking this function, a scaling process is triggered, which prompts the infrastructure to provision supplementary resources to effectively manage the increased workload. This function plays a crucial role in ensuring that the application can effectively handle elevated traffic and sustain optimal performance levels during peak periods.

First Lambda function for Scale-up (figure 22):



```
1 exports.handler = async (event) => {
2   // Your code for scaling up the infrastructure goes here
3   // This function can trigger auto-scaling groups, add more EC2 instances, or allocate additional resources based on specific conditions
4   // Example: Increase the desired capacity of an Auto Scaling Group by 2 instances
5
6   const AWS = require('aws-sdk');
7   const autoscaling = new AWS.AutoScaling();
8
9   const params = {
10    AutoScalingGroupName: 'js-app-auto-scaling-group',
11    DesiredCapacity: 2
12  };
13
14  try {
15    await autoscaling.setDesiredCapacity(params).promise();
16    console.log('Scaling up the infrastructure...');
17    return {
18      statusCode: 200,
19      body: 'Infrastructure scaled up successfully'
20    };
21  } catch (error) {
22    console.error('Error scaling up the infrastructure:', error);
23    return {
24      statusCode: 500,
25      body: 'Error scaling up the infrastructure'
26    };
27  }
28 }
29
```

Figure 22 Scale-up code for Lambda



```
Execution results
Test Event Name
test
Response
{
  "statusCode": 500,
  "body": " scaling up the infrastructure"
}
Function Logs
START RequestId: 19561141-4bb3-4ef5-9767-e1dd92ba67d5 Version: $LATEST
```

Figure 23 Succeeded test Scale-up function

## Second Function: Scale-Down Task





The purpose of this function is to automatically scale down the infrastructure based on a predefined threshold, such as decreased traffic or workload demand.

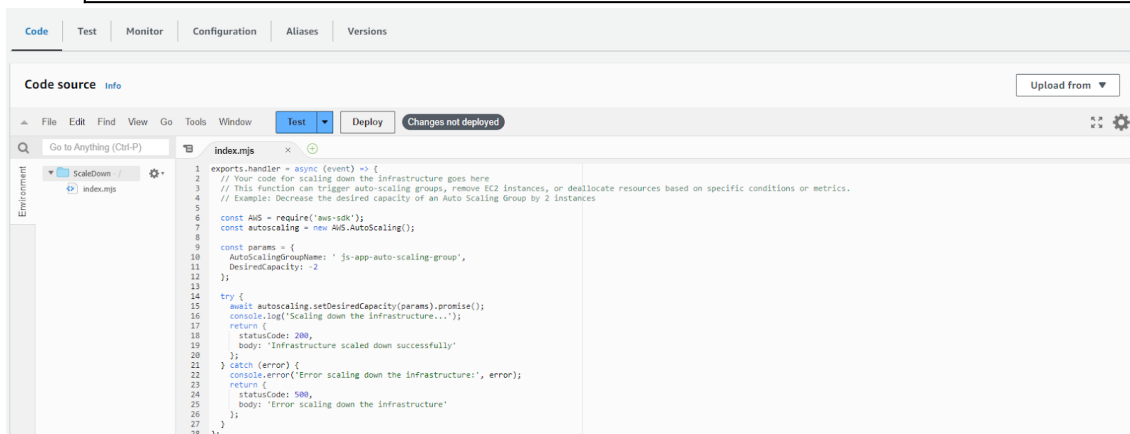
The function code (figure 23):

```
exports.handler = async (event) => {

  const AWS = require('aws-sdk');
  const autoscaling = new AWS.AutoScaling();

  const params = {
    AutoScalingGroupName: ' js-app-auto-scaling-group',
    DesiredCapacity: -2
  };

  try {
    await autoscaling.setDesiredCapacity(params).promise();
    console.log('Scaling down the infrastructure...');
    return {
      statusCode: 200,
      body: 'Infrastructure scaled down successfully'
    };
  } catch (error) {
    console.error('Error scaling down the infrastructure:',
error);
    return {
      statusCode: 500,
      body: 'Error scaling down the infrastructure'
    };
  }
};
```



The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. Below the tabs, there's a 'Code source' section with an 'Info' link and an 'Upload from' dropdown. The main area displays the code for the 'ScaleDown' function, with a file explorer on the left showing 'index.mjs'. The code is as follows:

```
1 exports.handler = async (event) => {
2   // Your code for scaling down the infrastructure goes here
3   // This function can trigger auto-scaling groups, remove EC2 instances, or deallocate resources based on specific conditions or metrics.
4   // Example: Decrease the desired capacity of an Auto Scaling Group by 2 instances
5
6   const AWS = require('aws-sdk');
7   const autoscaling = new AWS.AutoScaling();
8
9   const params = {
10    AutoScalingGroupName: ' js-app-auto-scaling-group',
11    DesiredCapacity: -2
12  };
13
14  try {
15    await autoscaling.setDesiredCapacity(params).promise();
16    console.log('Scaling down the infrastructure...');
17    return {
18      statusCode: 200,
19      body: 'Infrastructure scaled down successfully'
20    };
21  } catch (error) {
22    console.error('Error scaling down the infrastructure:', error);
23    return {
24      statusCode: 500,
25      body: 'Error scaling down the infrastructure'
26    };
27  }
28};
```

Figure 24 Scale-down code for Lambda

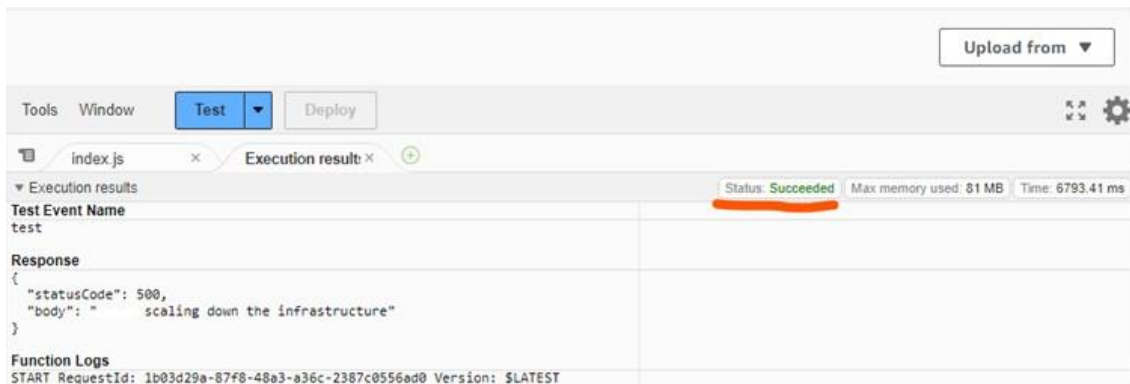


Figure 25 Succeeded test Scale-down function

The objective of the "Scale-Down" feature is to automate the process of downsizing the infrastructure, thereby minimizing resource consumption and reducing costs. This functionality is typically employed in situations where there is a decrease in demand or workload, necessitating the release of surplus resources to optimize operational efficiency.

The precise operations carried out by the "Scale-Down" function can vary depending on the specific implementation, but generally, it encompasses the following tasks:

1. Acquiring the current state and metrics of the infrastructure components, such as auto-scaling groups, EC2 instances, or other relevant resources.
2. Analyzing workload or demand patterns to assess the need for scaling down.
3. Calculating the optimal number of resources or capacity to be reduced based on predefined rules or thresholds.
4. Triggering the necessary actions to downscale the infrastructure, such as terminating EC2 instances, removing resources from load balancers, or adjusting auto-scaling group configurations.
5. Monitoring and verifying the successful completion of the scaling down process.
6. Providing appropriate feedback or a response indicating the outcome, whether it is a success or failure, of the scaling down operation.

By employing the "Scale Down" function, organizations can effectively optimize their resource allocation, mitigate unnecessary costs, and ensure that their infrastructure aligns with the actual demand or workload at any given time.

**Third Function: Monitor Resource Usage**

The Purpose of this function is Monitors the resource usage of the infrastructure and provides insights for better scaling decisions.

The function code (figure 26):

```

exports.handler = async (event) => {

  const AWS = require('aws-sdk');
  const cloudwatch = new AWS.CloudWatch();

  const params = {
    Namespace: 'AWS/EC2',
    MetricName: 'CPUUtilization',
    Dimensions: [
      {
        Name: 'InstanceId',
        Value: 'i-07446e304954ed261'
      }
    ],
    Statistics: ['Average'],
    Period: 300,
    StartTime: new Date(Date.now() - 300000), // Start time 5
minutes ago
    EndTime: new Date()
  };

  try {

    const data = await
cloudwatch.getMetricStatistics(params).promise();
    const averageCPUUtilization = data.Datapoints[0].Average;
    const threshold = 70; // Set your desired CPU threshold for
scaling

    if (averageCPUUtilization > threshold) {
      console.log('CPU utilization exceeds the threshold.
Scaling up the infrastructure...');
      // Trigger the Scale Up function or perform scaling
actions here
    } else {
      console.log('CPU utilization is below the threshold. No
scaling required.');
```

```

1 exports.handler = async (event) => {
2
3   const AWS = require('aws-sdk');
4   const cloudwatch = new AWS.CloudWatch();
5
6   const params = {
7     Namespace: 'AWS/EC2',
8     MetricName: 'CPUUtilization',
9     Dimensions: [
10      {
11        Name: 'InstanceId',
12        Value: 'i-07446e304954ed261'
13      }
14    ],
15    Statistics: ['Average'],
16    Period: 300,
17    StartTime: new Date(Date.now() - 300000), // Start time 5 minutes ago
18    EndTime: new Date()
19  };
20
21  try {
22    const data = await cloudwatch.getMetricStatistics(params).promise();
23    const averageCPUUtilization = data.DataPoints[0].Average;
24    const threshold = 70; // Set your desired CPU threshold for scaling
25
26    if (averageCPUUtilization > threshold) {
27      console.log('CPU utilization exceeds the threshold. Scaling up the infrastructure...');
28      // Trigger the Scale Up function or perform scaling actions here
29    } else {
30      console.log('CPU utilization is below the threshold. No scaling required.');
```

Figure 26 Monitor resource usage code for Lambda

Execution results: Status: Succeeded Max memory used: \$1 MB Time: 6893.71 ms

Test Event Name: test

Response:

```

{
  "statusCode":
  "body": monitoring resource usage
}
```

Function Logs:

```

START RequestId: ed0bcc40-fb7d-4b21-a829-cfaf774460ca Version: $LATEST
2023-06-22T21:10:18.779Z ed0bcc40-fb7d-4b21-a829-cfaf774460ca ERROR Error monitoring resource usage: TypeError: Cannot read property 'Av
at Runtime.exports.handler (/var/task/index.js:26:54)
at processTicksAndRejections (internal/process/task_queues.js:95:5)
END RequestId: ed0bcc40-fb7d-4b21-a829-cfaf774460ca
REPORT RequestId: ed0bcc40-fb7d-4b21-a829-cfaf774460ca Duration: 6893.71 ms Billed Duration: 6894 ms Memory Size: 128 MB Max Memory Used:
```

Request ID: ed0bcc40-fb7d-4b21-a829-cfaf774460ca

Figure 27 Succeeded test Monitor resource usage function

The primary objective of the "Resource Usage Monitoring" function is to systematically gather and track data on resource utilization for specific metrics within your infrastructure. This function enables the continuous monitoring of resource usage, empowering informed decision-making based on the comprehensive data collected.

Here are some distinctive characteristics of the "Monitor Resource Usage" function:

- Leveraging the AWS SDK, this function interacts with AWS services, primarily the CloudWatch service, to facilitate the collection and monitoring of resource usage data.
- It defines crucial parameters for the CloudWatch getMetricStatistics API call, including the namespace, metric name, dimensions, statistics, and time range.
- By employing the defined parameters, the function initiates an asynchronous API call to CloudWatch, retrieving the resource usage data.



- Extracting the pertinent information from the response, such as the metric value or aggregated statistics, is an essential step performed by the function.
- Depending on specific requirements, the function can execute diverse operations on the collected data. Examples include logging the data, storing it in a database, generating reports, or triggering actions based on predetermined conditions.
- Furthermore, the function can be scheduled to run at regular intervals using AWS CloudWatch Events or a similar scheduling mechanism. This feature ensures continuous monitoring of resource usage, allowing for insights into trends and patterns over time.

By implementing the "Monitor Resource Usage" function, organizations can proactively track and manage their resource utilization, enabling effective resource allocation, cost optimization, and the ability to respond promptly to changes in demand or workload.

To sum up, we could create more Lambda functions to make our application more serverless, elastic and easy to manage in case of errors or overloads.

### 3. Cognito

One of the most important steps in developing any application where we have a user Database with their personal data, is the security part. We have to implement the way they create accounts, authentication part, user management and synchronization. Thanks to Amazon Cognito it resolves all these needs and could cover most of the requirements. As it was described before, it has two main components: user and identity pools.

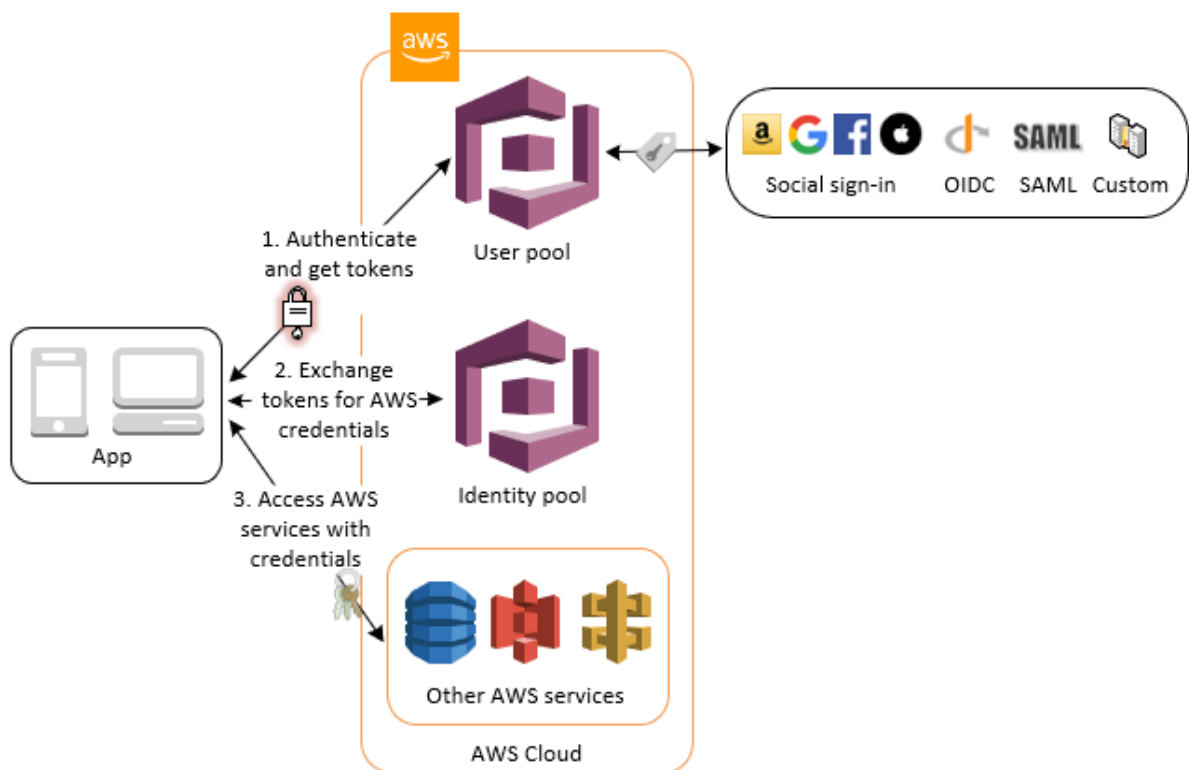


Figure 28 Access AWS services with a user pool and an identity pool [32]

As shown in figure 28, once a user pool authentication process is successfully completed, Amazon Cognito will provide tokens for the user pool associated with the application. These tokens serve as credentials and can be utilized to exchange for temporary access to a diverse range of AWS services through an identity pool. This robust mechanism enables applications to securely interact with AWS resources and leverage the extensive functionality offered by various services. By effectively utilizing these tokens, developers can ensure the secure and efficient utilization of AWS resources within their applications [22].

Firstly, we need to create the user pool:

- 1) Enter a name
- 2) In the attributes tab we configure how the users will sign in. It has 2 variants:
  - a) Use a username (Optionally, we could allow signing in with verified email, phone and preferred username, every option could be selected)
  - b) Use an email/phone (We could select to signing in with email or phone only, or both)

Also, we could choose which field in the sign-up form is required, for example: address, name, surname, birthday, gender, etc.

- 3) The next part is policies. Here we set a password length, if the special characters/numbers/uppercase letter/lowercase letter are required. We could allow users to create accounts themselves, or only administrators could do that. The last thing to set is the number of days when the temporary password expires.
- 4) The possibility to enable MFA (Multi-Factor Authentication) and make it optional or required. Configure the way how users will be able to recover their accounts: by email, phone or only by contacting the administrator. For verification or resetting an old password, we could choose where the code would be sent (email, phone (additional costs would be charged) or no verification).
- 5) In the next part, we could customize our email/phone message
- 6) We could put tags
- 7) Remember user's devices
- 8) App clients
- 9) Triggers

After the whole configuration, we will see a summary tab with all settings that we set, and we could modify any of them or just create it.



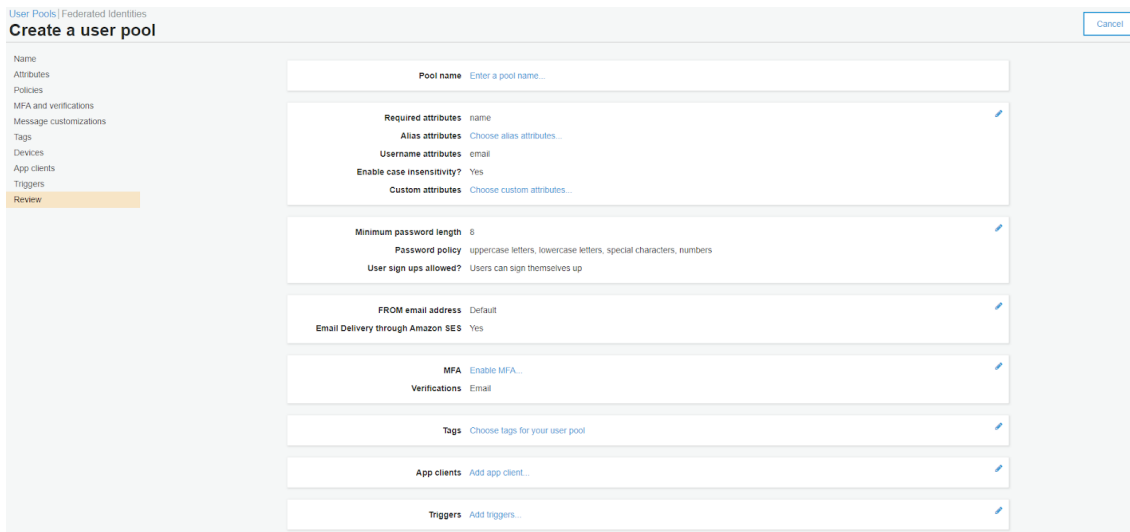


Figure 29 New user pool

To create an Identity pool, we need to complete 2 steps:

- 1) Enter a pool name, we could enable the access to the pool for unauthorized identities, choose authentication flow settings between enhanced or basic and authentication providers
- 2) Set permissions

Finally, we will get the next pool:

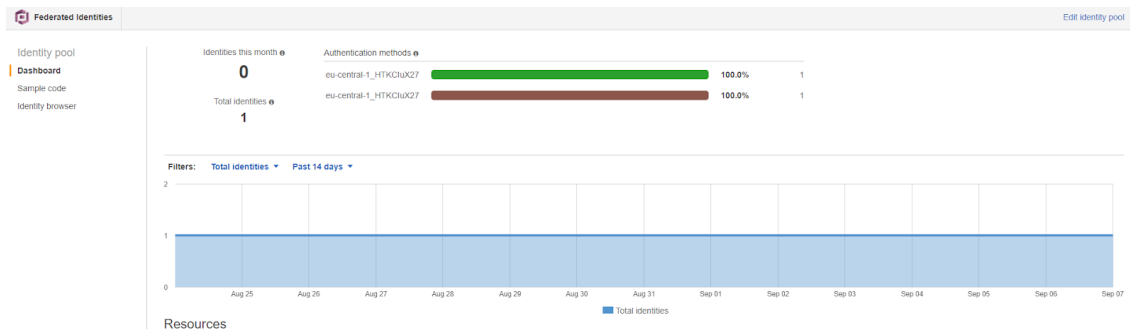


Figure 30 Identity pool

## 4. Updating

After the deployment of the application on AWS with Amplify, we could create our first canary using CloudWatch to monitor, analyze, and fine-tune the application during the release process (figure 31).

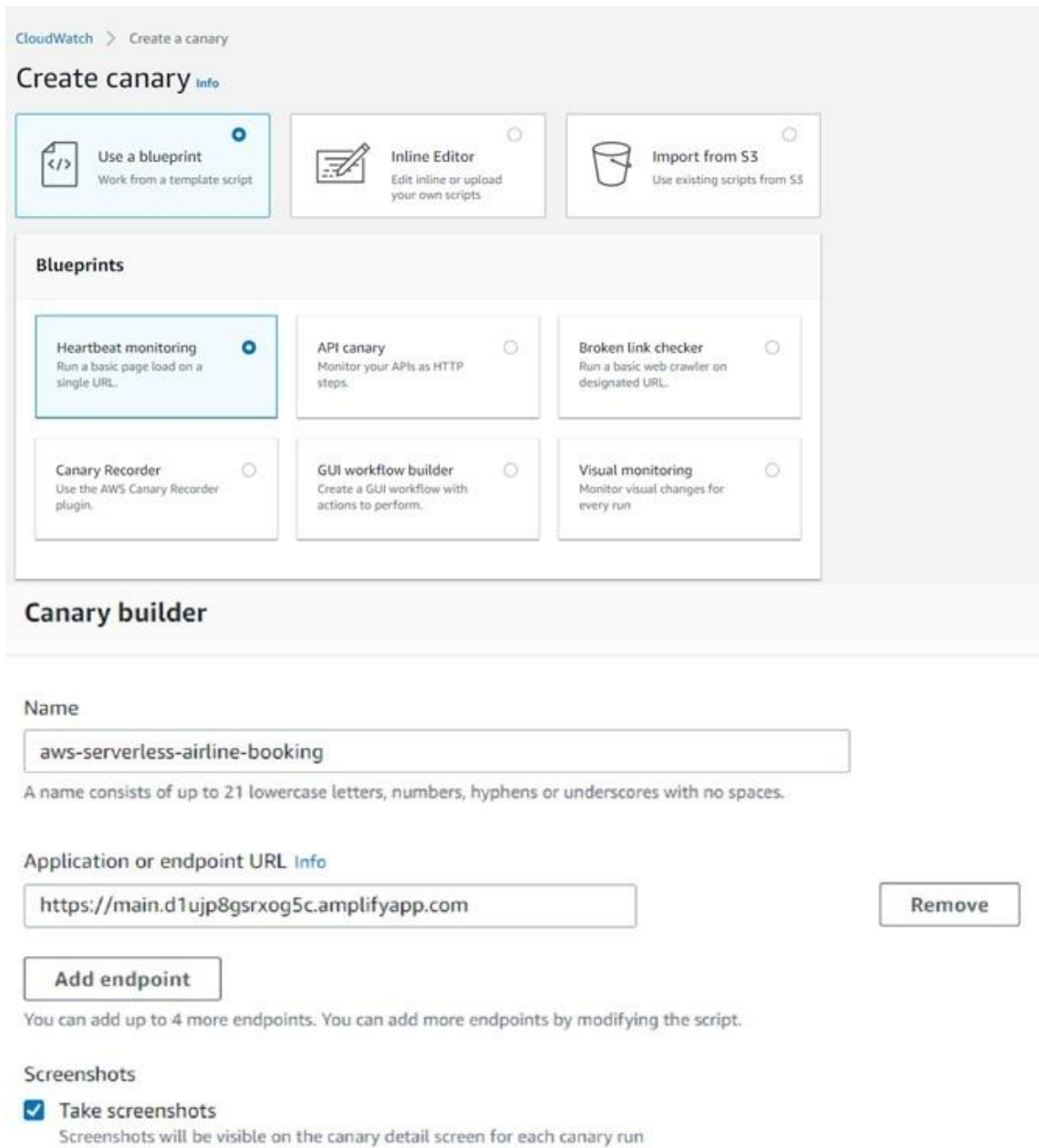


Figure 31 Canary creation

Monitoring plays a pivotal role in the success of a Canary release strategy, ensuring that the new version meets expectations while safeguarding the user experience. Below are unique monitoring practices to consider:

- **Comprehensive Performance Metrics:** Continuously monitor essential performance metrics during the canary stage, including response times, error rates, CPU and memory utilization, network latency, and any other relevant indicators specific to your application. These metrics provide insights into the overall health and efficiency of the new version.
- **In-Depth Analysis of Logs and Traces:** Dive into the logs and traces generated by the canary stage, leveraging powerful tools like AWS X-Ray, to uncover any errors,





exceptions, or performance bottlenecks. This detailed analysis sheds light on the root causes of issues and helps optimize the system's performance.

- **Intelligent Alerting and Proactive Notifications:** Establish robust alerting mechanisms that promptly notify you of any unusual behavior or breaches of predefined thresholds. This allows for immediate action and facilitates swift rollbacks if necessary, minimizing the impact on users. Proactive notifications ensure that you stay informed and can respond swiftly to any potential issues.
- **User-Centric Feedback Collection:** Solicit feedback from a select group of users directed to the canary stage, employing various methods such as surveys, feedback forms, or direct communication channels. This user-centric approach helps identify any usability issues or bugs that may have eluded automated monitoring, providing invaluable insights for improvement.

By implementing these monitoring practices, you can confidently navigate the intricacies of the Canary release strategy, ensuring the successful introduction of new features while maintaining an impeccable user experience.

In our case, we can monitor the app through an AWS service called CloudWatch as shown below in figure 32 and 33:

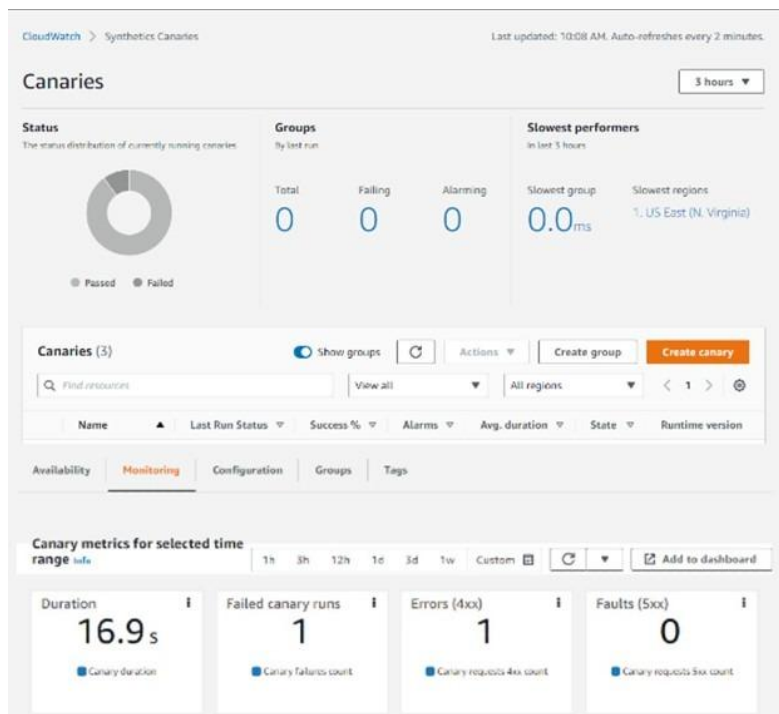


Figure 32 Canary monitoring

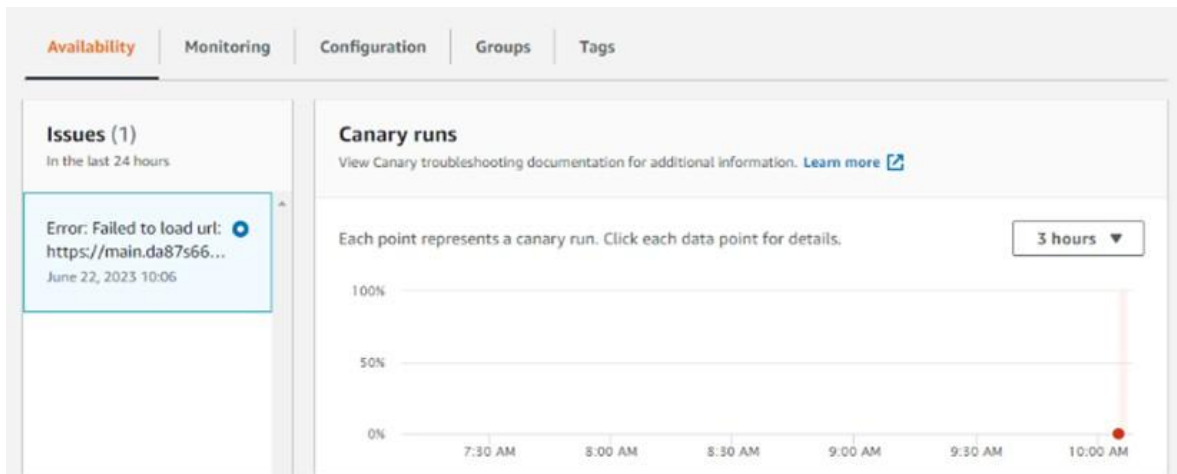


Figure 33 Canary runs

To deploy the JavaScript application using a Blue/green deployment strategy [20] the following AWS services were used:

- AWS EC2
- AWS CodeDeploy
- AWS IAM
- Github

- 1) Make a clone of the application repository on the local machine
- 2) Create an IAM role from AWS console and give it the “AmazonEC2RoleforAWSCodeDeployLimited” policy (figure 34):

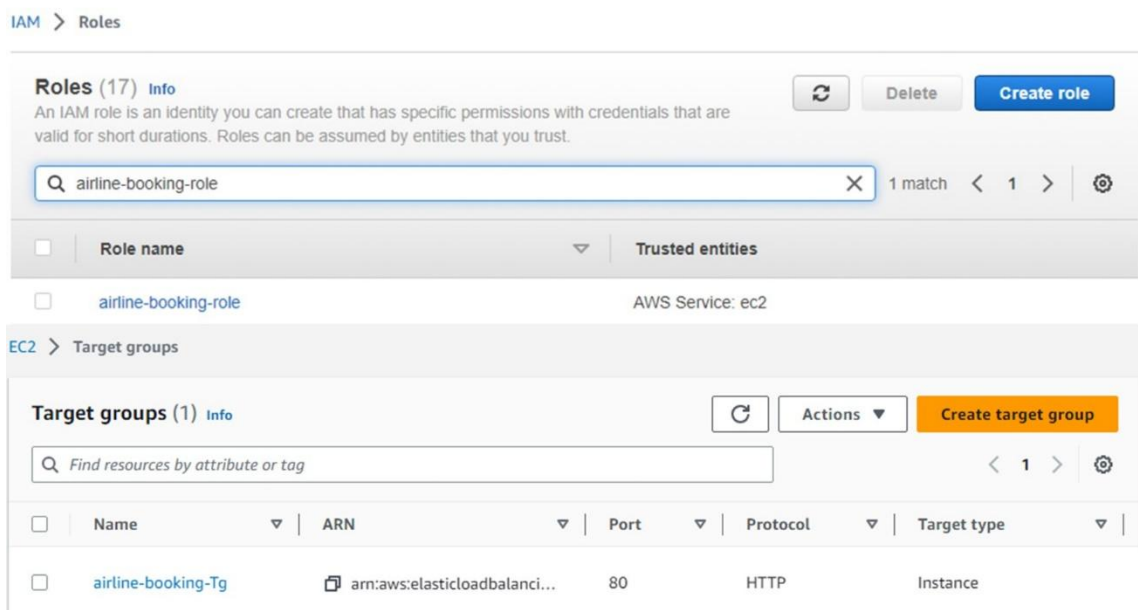


Figure 34 IAM role for Blue/Green deployment

- 3) Elastic load balancing attached to the previous group (figure 35):

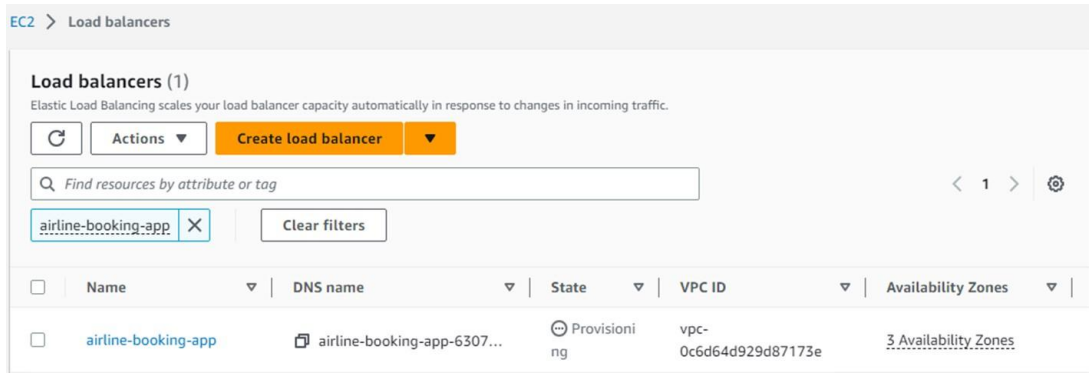


Figure 35 ELS

- 4) Create a launch configuration for Autoscaling (figure 36):

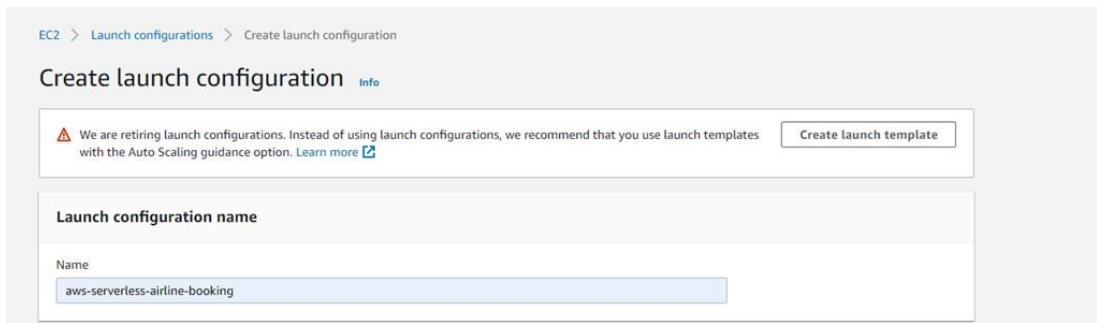


Figure 36 Launch configuration

- 5) Create an Auto Scaling group with the previous launch configuration (figure 37):

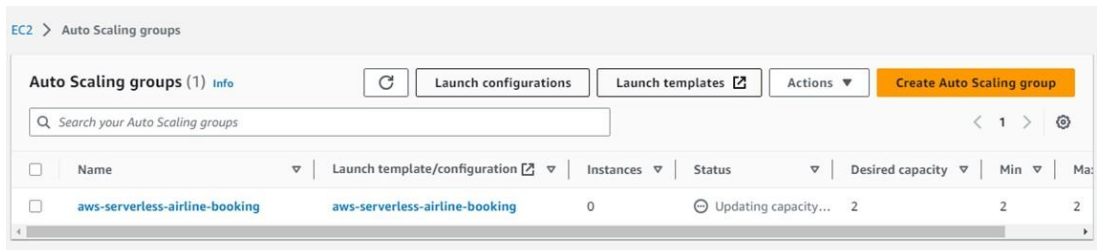


Figure 37 Auto Scaling group

- 6) Create an application on CodeDeploy (figure 38):

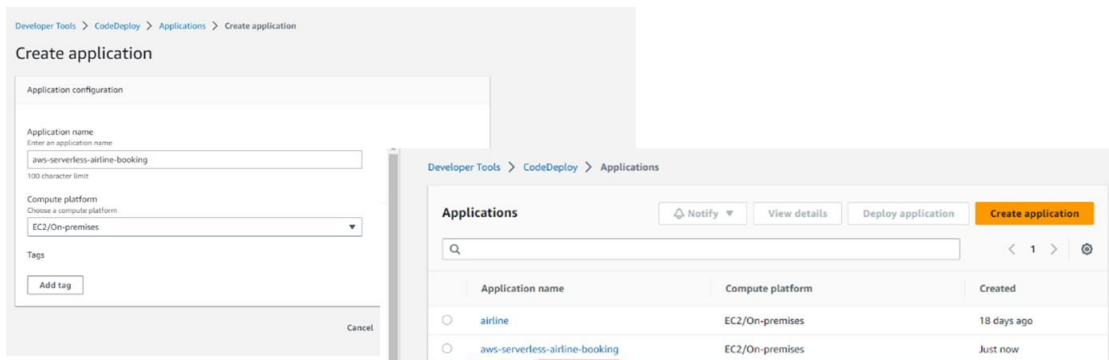


Figure 38 CodeDeploy creation

- 7) Create a deployment group on the code to deploy the application and choose Blue/green Deployment (figure 39):

The screenshot shows the 'Service role' section with a search bar containing 'arn:aws:iam::987174238652:role/CodeDeploy-airline-booking'. Below it, the 'Deployment type' section offers two options: 'In-place' and 'Blue/green'. The 'Blue/green' option is selected and highlighted in blue. The description for 'Blue/green' states: 'Replaces the instances in the deployment group with new instances and deploys the latest application revision to them. After instances in the replacement environment are registered with a load balancer, instances from the original environment are deregistered and can be terminated.'

Figure 39 Deployment group creation

- 8) Create a Deployment on CodeDeploy -> Deployments (figure 40):

The screenshot shows the 'Deployment details' for a deployment named 'd-LLV51UA50'. The details are as follows:

Application	Deployment ID	Status
aws-serverless-airline-booking	d-LLV51UA50	Created
Deployment configuration	Deployment group	Initiated by
CodeDeployDefault.AllAtOnce	aws-serverless-airline-booking-DG	User action
Deployment description	First deployment	

Figure 40 New deploy on CodeDeploy

After Deployment is created 2 instances are automatically launched by the autoscaling group that is configured with CodeDeploy as shown in figure 41:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
aws-serverless...	i-07ccc4eb466629c11	Running	t2.micro	Initializing	No alarms	us-east-1b
aws-serverless...	i-0eb203d9612828f1c	Running	t2.micro	Initializing	No alarms	us-east-1a

Figure 41 Active instances in CodeDeploy

Now the app is hosted with Blue/Green Deployment.



The Blue environment after deployment started (figure 42):

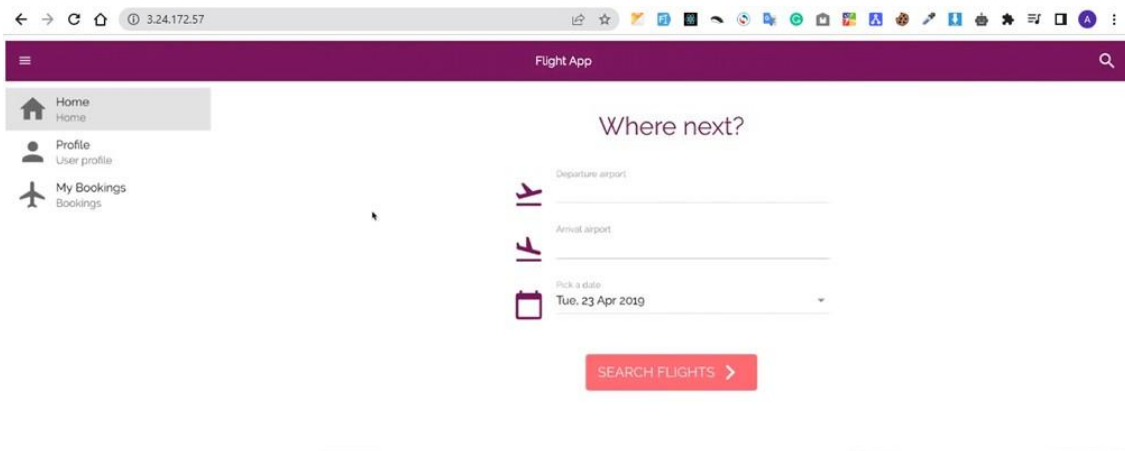


Figure 42 Blue environment

In figure 43 shows the Green Environment Before testing (inactive):

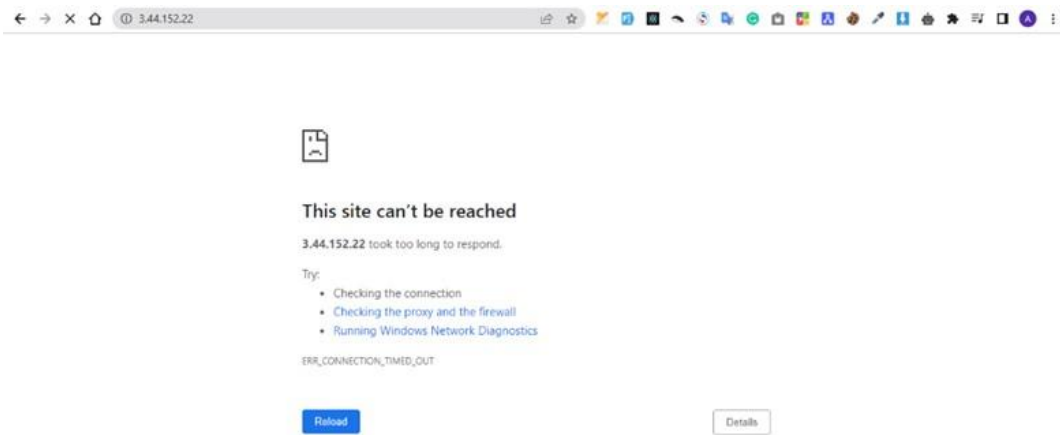


Figure 43 Green environment before deployment

After the green environment has been tested and verified, the traffic switched to this environment and now it is active (figure 44):

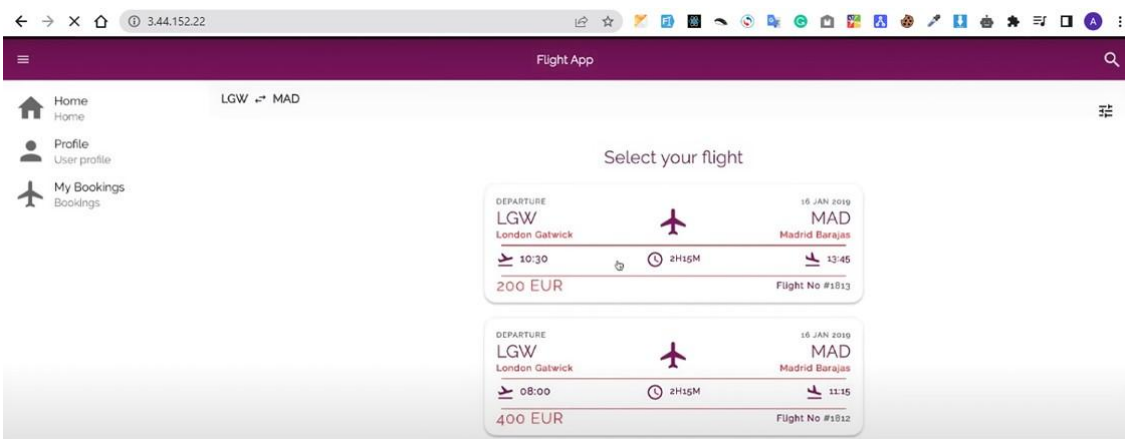


Figure 44 Green environment after

Blue/Green deployment in AWS is a robust approach that ensures smooth and risk-free updates to applications. It involves maintaining two identical environments, Blue and Green, where the Blue environment represents the existing stable version while the Green environment hosts the updated version. By directing traffic to the Green environment after a successful update, any potential issues can be detected without impacting end users. If issues arise, traffic can be immediately redirected back to the stable Blue environment. This deployment strategy provides increased reliability, minimizes downtime, and allows for easy rollbacks in case of unexpected problems. With Blue/Green deployment in AWS, developers can confidently deliver updates while maintaining a seamless user experience.

## 5. AWS X-RAY

During the deployment of a JavaScript application with Amplify by Canary release strategy it has an option named X-Ray trace on AWS CloudWatch that integrates directly with the app deployed as shown in figure 45.

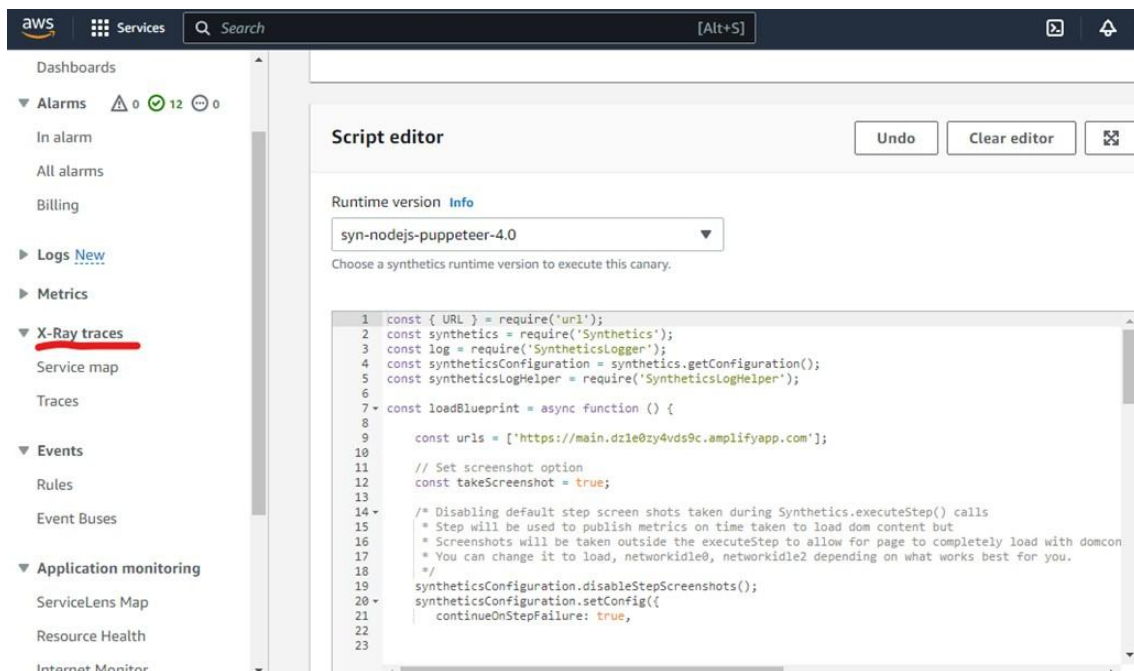


Figure 45 X-RAY traces option

The presence of X-Ray traces on the CloudWatch dashboard signifies that the trace data produced by the application's integration with AWS X-Ray is being gathered and exhibited within the CloudWatch service.

When X-Ray traces are visible on the CloudWatch dashboard, it indicates that the trace data, which is generated through the application's seamless integration with AWS X-Ray, is actively collected and presented within the CloudWatch service. This integration allows for a comprehensive view of the application's tracing information, providing valuable insights into its performance and behavior. By leveraging this combined functionality, developers and operators can gain a holistic understanding of



their application's execution and diagnose any potential issues or bottlenecks efficiently [21].

When it comes to integrating AWS X-Ray into an application, the process typically involves the following steps:

1. **Application Instrumentation:** Begin by incorporating the X-Ray SDK or library into your application code. This entails adding the necessary dependencies and initializing the X-Ray client to enable tracing capabilities.
2. **Sampling Rules Configuration:** X-Ray provides the flexibility to define sampling rules, allowing you to control the amount of data collected. You can specify the percentage of requests to sample or establish specific conditions for capturing traces based on your requirements.
3. **Tracing Activation:** Activate X-Ray tracing for your application. This can be achieved through configuration settings or by utilizing environment variables that trigger the tracing functionality.
4. **Traces Collection and Analysis:** Deploy the instrumented application and start generating traces. Utilize the AWS X-Ray console or APIs to collect, view, and analyze the generated traces. This enables you to identify potential issues, gain valuable insights into your application's performance, and make informed optimizations.

By following these steps, you can seamlessly integrate AWS X-Ray into your application, enabling comprehensive tracing capabilities and empowering you to monitor and improve your application's performance and reliability.

Installing AWS X-Ray SDK locally on computer (figures 46 and 47):

```
C:\WINDOWS\system32>cd aspnetcore-deployment-samples-main
C:\Windows\System32\aspnetcore-deployment-samples-main>npm install aws-xray-sdk
added 41 packages in 9s
C:\Windows\System32\aspnetcore-deployment-samples-main>
```

*Figure 46 X-RAY SDK installation*

```

48     spinnerSize: 200 // px
49   });
50
51   Vue.config.productionTip = false;
52
53   new Vue({
54     router,
55     store,
56     render: h => h(App)
57   }).$mount("#app");
58
59
60
61   // Import the required modules
62   const AWSXRay = require('aws-xray-sdk');
63   const AWS = AWSXRay.captureAWS(require('aws-sdk'));
64
65   // Instrument your code to capture traces
66   // Example: Capture an S3 request
67   const s3 = new AWS.S3();
68   AWSXRay.captureAWSClient(s3);
69
70   s3.getObject({ Bucket: 'my-bucket', Key: 'my-object' }, (err, data) => {
71     // Handle response
72   });

```

Figure 47 X-RAY implementation

Then it is necessary to check if everything is working, go to CloudWatch Console -> AWS X-RAY -> Traces.

As we can see on Figure 48 for the “HTTP. Method == GET” we have 9 traces.

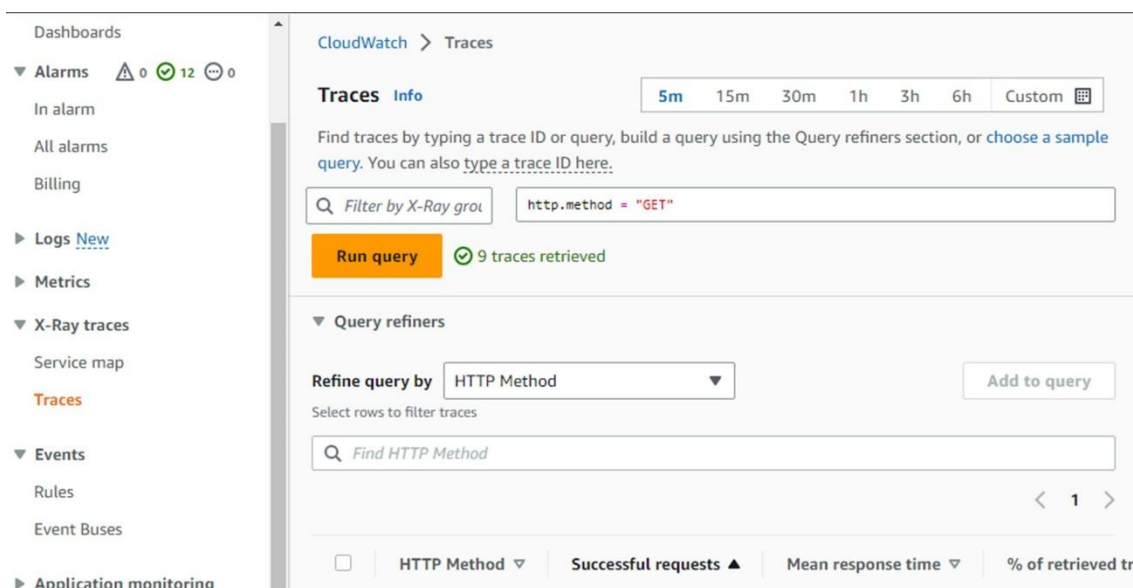


Figure 48 X-RAY traces

Here's an overview of the capabilities and key features at our disposal:

- Request Tracing and Monitoring: AWS X-Ray empowers us to trace and monitor the flow of requests within our application. By capturing data about these requests, it offers valuable insights into latency, performance, and behavior of individual components.
- Application Instrumentation: To utilize AWS X-Ray, the application code needs to be instrumented by integrating the AWS X-Ray SDK. This SDK provides libraries tailored for different programming languages, facilitating the capture and transmission of trace data to AWS X-Ray.





- **Trace Visualization:** AWS X-Ray furnishes a visual representation of the captured traces, allowing us to comprehend the path and timing of requests across various services and components. This visualization aids in identifying bottlenecks, performance issues, and dependencies within our application.
- **Service Maps:** AWS X-Ray generates service maps that depict the relationships and dependencies between different services within our application. These service maps facilitate a comprehensive understanding of the application's architecture and component interactions.
- **Error Analysis:** AWS X-Ray captures errors and exceptions occurring within our application and associates them with the corresponding traces. This seamless linkage enables easier identification and debugging of issues by providing detailed error information.
- **Performance Insights:** Leveraging AWS X-Ray, we can analyze the performance of our application and identify areas for optimization. It offers metrics and visualizations that shed light on the impact of different components on overall performance.
- **Integration with AWS Services:** AWS X-Ray seamlessly integrates with various AWS services, including AWS Lambda, Amazon EC2, and Amazon API Gateway. This integration enables tracing requests across different services, providing end-to-end insights into our application's behavior.
- **Debugging and Troubleshooting:** AWS X-Ray simplifies the debugging and troubleshooting process by offering detailed information about request flow and individual component performance. This feature expedites issue identification and resolution. Overall, AWS X-Ray equips developers with invaluable insights into their application's behavior and performance within distributed environments. It facilitates performance optimization, efficient troubleshooting, and enhanced user experience.

# Chapter 5. Conclusions and Future Work

---

The main objective of this project was to show how any application/product could use the power of cloud solutions, techniques, tools and Amazon Web Services. It demonstrates how flexible the AWS could be and how adaptable for specific project requirements, no matter how big it is. It gives an opportunity to have a scalable infrastructure, efficient resource management and seamless integration of various services.

By utilizing cloud solutions and AWS services, this project showcases the ability to dynamically scale resources based on demand. This scalability ensures that the application can handle fluctuations in user traffic, optimize operations and workload without compromising performance or reduce costs and deliver stable high-quality service.

## 1. Achievement of Objectives

At the beginning of the project there were mentioned the goals to achieve during this work. The first part was to deploy the application into the cloud using serverless framework and AWS services. In this step the first difficulties and errors began to appear, which required further deeper study of this field and also search for information about the services that were used. Due to the fact that cloud solutions and serverless techniques have only recently become so popular, it was time-consuming and had an effect on problem-solving and information search because of the small amount of information.

The next part was to discover how to scale the application for the different scenarios. The most common problem with any product/application is the user traffic and it is impossible to predict all situations that could happen and when but special functions for different cases could be prepared. That is why scalable Lambda functions were implemented for the cases when the traffic suddenly increased or decreased. Thanks to these functions it is possible to automate server resources and increase or decrease them in different situations, and this could reduce costs, increase availability of the application and its reliability. Also, a monitor resource usage function was added for better scaling decisions and correct operation of the application in real time.

Moreover, it was important to connect a service that could monitor in real time the correct operation of the entire application, logs, errors, server load, etc. For these needs



AWS X-RAY service has been chosen but the lack of information, guides and low popularity caused a fairly large number of problems and errors during the implementation. This part showed a flexible monitoring system that could help to maintain the system up, future updates, searching the bottlenecks and bug detecting.

An important part of any application or product is its security. The security part of this project describes the Cognito service from AWS. It shows how easily and flexibly the part of authentication, registration and user pool management could be configured. There were several steps, rules and requirements for sign-in and sign-up that represent how powerful it is.

Last but not least, the step for the system updates was introduced using Canary release and Blue/Green deployment. It showed how these techniques were implemented in AWS using different services such as CodeDeploy, CloudWatch, etc. That gives a possibility to maintain the working copy of the application during the testing of an updated one.

## **2. Future work**

This project has a lot of potential for future work and there are a lot of things that can be added, configured and implemented. Amazon Web Services offers more than 200 fully featured services in different domains, including compute, storage, databases, networking, analytics, machine learning, AI, security, and more.

Based on already implemented services in this work, more Lambda functions could be created for the application for more cases and situations. Thanks to them it is possible to automate resolving the most common problems and issues, reduce costs and increase scalability and availability.

For the X-RAY configuration, service maps could be created for a better visualization, integrating it with the other services such as AWS Lambda, AWS Elastic Beanstalk, Amazon EC2 or third-party frameworks and tracing libraries. Spend more time on analyzing captured data by X-RAY to identify bottlenecks, bugs and anomalies.

For the deployment part, other strategies could be implemented, such as Hybrid Deployment, Rolling Deployment, All-at-Once Deployment, depending on special requirements or factors.

Regarding the security layer for the application, AWS offers other services that could be used with Cognito that increase the level of security. For example Amazon Macie uses machine learning to automatically discover, classify, and protect sensitive data or AWS Shield that protects from DDoS (Distributed Denial of Service) attacks.

# Bibliography

---

- [1] “What is serverless?,” 11 May 2022. [Online]. Available:  
<https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless>.
- [2] T. F. Richard and R. T. Freeman, Building Serverless Microservices in Python: A Complete Guide to Building, Testing, and Deploying Microservices Using Serverless Computing on AWS, Packt Publishing, 2019.
- [3] S. Maarek, AWS Lambda and the Serverless Framework: Hands-on Learning!, Packt Publishing, 2018.
- [4] A. W. Services. [Online]. Available:  
<https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>.
- [5] A. W. Services. [Online]. Available: <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>.
- [6] M. Mamdouh, 04 2023. [Online]. Available: <https://dev.to/aws-builders/aws-amplify-the-one-stop-shop-for-mobile-and-full-stack-development-412o>.
- [7] A. Y. Ogun, 05 2022. [Online]. Available: <https://awstip.com/aws-x-ray-how-does-it-work-a3c9a8f189a6>.
- [8] V. ATHITHAN, 03 2020. [Online]. Available:  
<https://cloudacademy.com/blog/what-is-cognito-in-aws/>.
- [9] A. W. Services. [Online]. Available:  
<https://docs.aws.amazon.com/apigateway/latest/developerguide/canary-release.html>.
- [10] CodeFresh. [Online]. Available: <https://codefresh.io/learn/software-deployment/what-is-blue-green-deployment/>.
- [11] T. Fernandez, 07 2022. [Online]. Available: <https://semaphoreci.com/blog/what-is-canary-deployment>.



- [12] A. W. Services. [Online]. Available: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-blue-green.html>.
- [13] C. ACHINGA, 09 2022. [Online]. Available: <https://cloudacademy.com/blog/aws-codedeploy-what-it-is-how-it-works/>.
- [14] “Intellipaat,” 05 2023. [Online]. Available: <https://intellipaat.com/blog/what-is-cloudwatch-in-aws/?US>.
- [15] A. Anthony, Mastering AWS Security: Create and maintain a secure cloud ecosystem, Packt, 2017.
- [16] K. Parmar, 2022. [Online]. Available: <https://www.elitechsystems.com/why-javascript-is-so-popular-programming-language/>.
- [17] A. W. Services. [Online]. Available: <https://aws.amazon.com/serverless/aws-sam/>.
- [18] Aws-samples, “GitHub,” [Online]. Available: [https://github.com/aws-samples/aws-serverless-airline-booking/blob/archive/docs/getting\\_started.md](https://github.com/aws-samples/aws-serverless-airline-booking/blob/archive/docs/getting_started.md).
- [19] A. W. Services. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html>.
- [20] A. W. Services, 09 2021. [Online]. Available: [https://d1.awsstatic.com/whitepapers/AWS\\_Blue\\_Green\\_Deployments.pdf](https://d1.awsstatic.com/whitepapers/AWS_Blue_Green_Deployments.pdf).
- [21] OpsRamp. [Online]. Available: <https://www.opsramp.com/guides/aws-monitoring-tool/aws-x-ray/>.
- [22] A. W. Services. [Online]. Available: <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.html>
- [23] A. W. Services. [Online]. Available: <https://docs.aws.amazon.com/lambda/latest/operatorguide/provisioned-scaling.html>

- [24] C.Le, 03 2021. [Online]. Available: <https://vticloud.io/en/gioi-thieu-dich-vu-aws-lambda-va-cach-cau-hinh-cho-nguoi-moi-bat-dau/>
- [25] D.Sato, 06 2014. [Online]. Available:  
<https://martinfowler.com/bliki/CanaryRelease.html>
- [26] A. W. Services. [Online]. Available:  
<https://aws.amazon.com/blogs/startups/how-to-use-blue-green-deployment-on-aws/>
- [27] A. W. Services. [Online]. Available:  
<https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html>
- [28] M. Ozkaya, 09 2022. [Online]. Available: <https://medium.com/aws-lambda-serverless-developer-guide-with-hands/amazon-cognito-main-features-user-pools-and-identity-pools-uses-cases-and-how-it-works-20fbe94b1905>
- [29] “coolestguidesontheplanet.”, 07 2014. [Online]. Available:  
<https://coolestguidesontheplanet.com/installing-nodejs-command-line-linux-osx/>
- [30] “aws-samples”. [Online]. Available:  
<https://github.com/aws-samples/aws-serverless-airline-booking/tree/archive>
- [31] A. W. Services. [Online]. Available:  
<https://docs.aws.amazon.com/lambda/latest/operatorguide/provisioned-scaling.html>
- [32] A. W. Services. [Online]. Available:  
<https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.html>



# Appendix. SDG

Relevance of the work to the Sustainable Development Goals (SDGs).

Sustainable Development Goals	High	Medium	Low	Not applicable
ODS 1. <b>No poverty.</b>				<b>X</b>
ODS 2. <b>Zero hunger.</b>				<b>X</b>
ODS 3. <b>Good health and well-being.</b>				<b>X</b>
ODS 4. <b>Quality education.</b>		<b>X</b>		
ODS 5. <b>Gender equality.</b>				<b>X</b>
ODS 6. <b>Clean water and sanitation.</b>				<b>X</b>
ODS 7. <b>Affordable and clean energy.</b>				<b>X</b>
ODS 8. <b>Decent work and economic growth.</b>	<b>X</b>			
ODS 9. <b>Industry, innovation and infrastructure.</b>	<b>X</b>			
ODS 10. <b>Reduced inequality.</b>				<b>X</b>
ODS 11. <b>Sustainable cities and communities.</b>				<b>X</b>
ODS 12. <b>Responsible production and consumption</b>		<b>X</b>		
ODS 13. <b>Climate action.</b>		<b>X</b>		<b>X</b>
ODS 14. <b>Life below water.</b>				<b>X</b>
ODS 15. <b>Life on land.</b>				<b>X</b>
ODS 16. <b>Peace, justice and strong institutions.</b>				<b>X</b>
ODS 17. <b>Partnerships for the goals.</b>				<b>X</b>

Consideration of the relationship of the TFG/TFM with the SDGs and with the most relevant SDG(s). This Final Degree Project has an impact on 5 of the 17 sustainable development objectives as shown in the table.

AWS plays a significant role in supporting the Sustainable Development Goal (SDG) of Quality Education through its diverse set of cloud services and educational initiatives. By leveraging AWS technologies, educational institutions and organizations can enhance the accessibility, affordability, and effectiveness of education, driving positive impact worldwide. One of the key contributions of AWS in achieving Quality Education is through its cloud infrastructure. AWS provides a reliable and scalable platform that enables educational institutions to deliver online learning experiences, virtual classrooms, and collaborative tools. This accessibility to technology fosters inclusive education, reaching learners from diverse backgrounds and geographical locations.

Apart from quality education it contributes significantly to Decent Work and Economic Growth by empowering businesses, entrepreneurs, and workers with the tools and technologies needed to drive innovation, productivity, and economic development. Through its cloud services and infrastructure, AWS enables businesses of all sizes to scale their operations, enhance efficiency, and reduce costs. This scalability promotes

business growth, job creation, and economic opportunities, particularly for small and medium-sized enterprises (SMEs) and startups. By providing on-demand computing resources, AWS empowers organizations to focus on their core competencies and innovation, creating a conducive environment for job creation and economic expansion.

For Industry, Innovation, and Infrastructure it has a high impact by providing the technological foundation and tools necessary for businesses and organizations to drive innovation, enhance infrastructure, and foster economic growth. Thanks to cloud services and technologies, AWS empowers businesses of all sizes to innovate and develop cutting-edge solutions. The scalability, flexibility, and cost-effectiveness of AWS services enable organizations to experiment, iterate, and bring new products and services to market faster. This culture of innovation promotes entrepreneurship, stimulates economic growth, and encourages the development of sustainable industries.

Apart from that, it supports the development of digital infrastructure by providing reliable, secure, and scalable cloud computing services. By leveraging AWS's infrastructure, businesses and organizations can build robust and resilient systems that support their operations and enable them to scale seamlessly. This infrastructure enhances the reliability and availability of critical services, contributing to the development of resilient and sustainable infrastructure.

AWS contributes to the achievement of the Sustainable Development Goal (SDG) of Responsible Production and Consumption by promoting sustainable practices, reducing waste, and enabling responsible consumption through its cloud services and initiatives. Helping businesses optimize their resource usage and reduce waste is one of examples how AWS supports responsible production. By leveraging AWS's cloud infrastructure, organizations can scale their computing resources according to demand, ensuring that they only consume the necessary amount of energy and computing power. This elasticity not only improves operational efficiency but also reduces the environmental impact associated with excessive resource consumption.

Climate Action is the last sustainable development objective that AWS has impacted on. By leveraging its cloud infrastructure and implementing various initiatives to drive environmental sustainability and combat climate change. AWS, one of the most widely used cloud computing platforms, is essential in assisting companies minimize their environmental effect and reduce their carbon footprint. Organizations could significantly decrease their energy use and related greenhouse gas emissions by moving to the AWS cloud. The energy efficiency of its data centers is enhanced using cutting-edge cooling and power management technology. They are also dedicated to using only renewable energy for their entire global infrastructure. AWS is actively converting its data centers to use clean and sustainable energy sources, such as wind and solar, through long-term power purchase agreements and investments in renewable energy initiatives. This dedication to renewable energy helps to reduce carbon emissions.

