SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

Project Proposal

CLOUD COMPUTING

Professor:
Emiliano Casalicchio

Students:
Andrea Di Marco
Jemuel Espiritu Fuentes
Michele Granatiero
{dimarco.1835169, fuentes.1803530, granatiero.1812623}@studenti.uniroma1.it

---

Academic Year 2022/2023

Contents

1 Introduction **2**

2 Implementation **3**

2.1 AWS Services 3

2.2 Other frameworks 3

3 Testing **4**

1 Introduction

Our intent is to design and develop an online *ticketing service* by leveraging on cloud technologies called TicketCloud. We aim to deploy a user friendly web application that can scale and support a large number of requests at the same time with no issues. Since the service is centered on an ever growing market, with millions of people wanting to take part in thousands of different events everyday, resiliency and scalability will be an important aspect in designing the software.
The system will require a user to create an account in order to log in our system and have access to most of the functionalities. The users will be able to browse various events, book the tickets they desire and eventually pay. The users will also be able to cancel their reservation if they desire to. The system will keep track of previous reservations for each user and might implement a recommending system.
The project will also deal with an important aspect of developing software which is the testing and validation of the entire cloud architecture.

2 Implementation

TicketCloud will be built on top of the Amazon Web Service infrastructure following a loosely-coupled and serverless architecture based on microservices.

2.1 AWS Services

1. **AWS Amplify** or **AWS Elastic Beanstalk** for the full stack development.
2. **Amazon S3**: scalable object storage. We will use this service to store the static resources of the website.
3. **DynamoDB**: scalable, NoSQL database service. We will rely on DynamoDB to store the user informations, ticket informations and event details of TicketCloud.
4. **AWS Lambda**: event-driven calculus service. It will be used to operate on the database with CRUD APIs.
5. **Amazon Cognito**: user authentication and access control service.
6. **Amazon API Gateway**: creation of REST APIs and Lambda Integration.
7. **AWS Well-Architected** and **AWS Trusted Advisor**, tools to facilitate decision making when choosing services.

2.2 Other frameworks

1. **Angular**: open source framework for web applications.

3 Testing

During the testing phase for TicketCloud we will different testing approaches to verify and validate the system.

1. **Automated Testing Environments**: An important benefit of running tests on AWS is the ability to automate them in various ways. We can create and manage test environments programmatically using the AWS APIs, CLI tools, or AWS SDKs. Tasks that require human intervention in classic environments like allocating a database can be fully automated using *AWS CloudFormation*.

2. **Load & Scalability Testing**: In Serverless architectures using AWS services such as *AWS Lambda* and *Amazon API Gateway* load testing can help identify custom code in Lambda functions that may not run efficiently as traffic scales up. This can be performed by a combination of ad-hoc *AWS Lambda* scripts and *AWS CloudWatch*. Tailor made scripts and benchmarking tools such as *JMeter*, *Locust* or *Artillery* will be used to test the elasticity of TicketCloud.

3. **User Testing**: The objective of user acceptance testing is to present the current release to a testing team representing the final user base, to determine if the project requirements and specification are met. When users can test the software earlier, they can spot conceptual weaknesses that have been introduced during the analysis phase, or clarify gray areas in the project requirements.

4. **Fault-tolerance Testing**: Specific test practices provide insights into how the system will handle component failures. AWS offers many options for building fault-tolerant systems. Some services are inherently fault-tolerant, for example *Amazon DynamoDB* and *Amazon CloudFront*. Other services still provide features that help architect fault-tolerant and highly available systems.

We will also try to perform **Cost Optimization** for our architecture, in order to limit the amount of superfluous expenses.