2 CSCI 5409

ADVANCE TOPICS IN CLOUD COMPUTING

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

TICKET BOOKING SYSTEM

Architecture Critical Analysis and Response

GROUP – 8 ALPHA TEAM

DHRUV DOSHI	**Dh722257@dal.ca**
KISHAN KAHODARIYA	**Ks805556@dal.ca**
VISHAL RAKESH JAISWAL	**Vs928999@dal.ca**

Contents

Contents of Figure

This report illustrates the design architecture of the application and details about the various cloud services used within the system.

ARCHITECTURE DIAGRAM

FLOW OF DATA

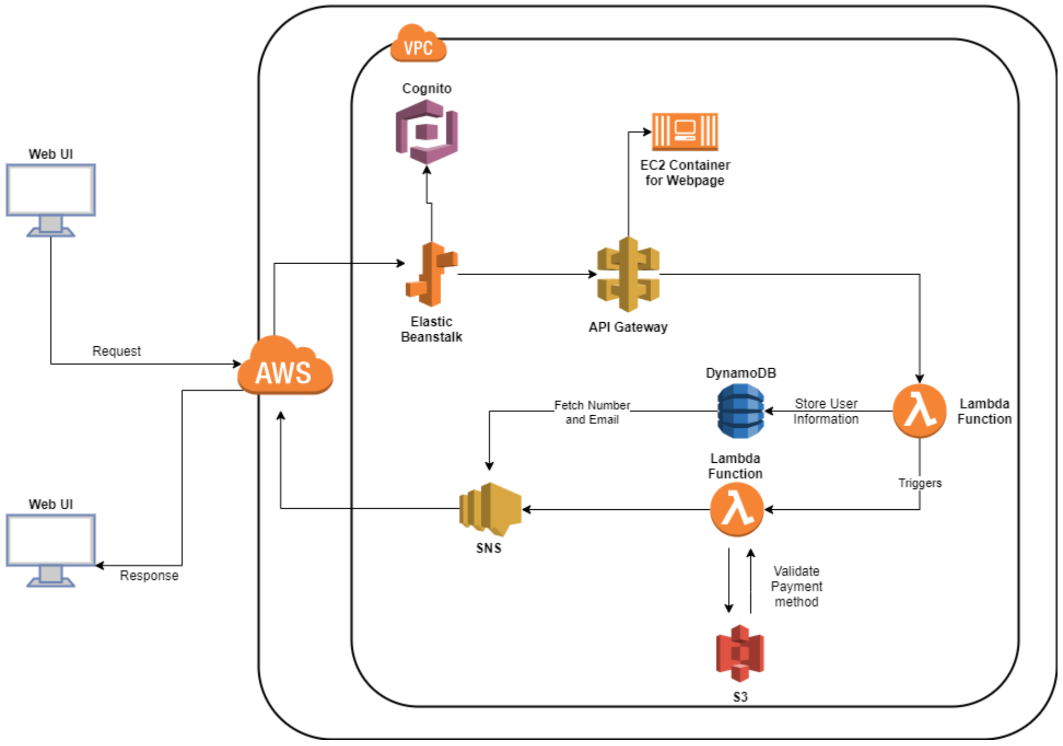


Figure 1 Design Architecture

COMPONENTS

All the components that make up the cloud software

Figure 1 accurately represents all the main component that will be used within the system and also the order in which it will be called along with their individual functionality. Given group needs all member of the group to operate on code, deploy and test their individual code and for that sole reason we have opted for Amazon Elastic Beanstalk which is allows you to manage all of your project's deployment, scaling of the project and also load balancing which is something we will need in the future [1]. Along with this, beanstalk also manages all services for all containers deployed and it allows you to control the type of request the system receive and forward it to next appropriate service. In order to use our system and book tickets, user needs to authenticate themselves using their email or userID and password. For such implementations, AWS cognito comes in handy as it allows you to add sign-in and signups for your web or mobile application [2]. In our project, using AWS Cognito, we will fetch the data from user using web and then verify that credentials against our database and check if the user is authentic or not and allow access to the system accordingly and if user is registering for the first time, then store those details in an S3 Bucket. Note that as an extra measure of security, AWS Secret Manager is also used to assign appropriate roles to each user at the time of sign up to avoid unnecessary access to a service which user is not supposed to gain access to [3]. API Gateway will manage all type of API calls amongst the services in our system [3]. Lambda functions are triggered based on incoming raw data and then events are launched based on their outcome [5]. At last, once the ticket is successfully booked and payment method is validated using another lambda function, system will invoke SNS service to send text or email-based notification to the user regarding successful booking onto their respecting phone number which will be fetched from DynamoDB [6].

IMPORTANT ASPECTS

Description of important aspects of the architecture, for example which components are key and may require high availability?

Based on our discussion and through readings, we think that higher availability of Elastic Beanstalk, Lambda and Cognito is essential for our cloud system. The reason is that as our system will be containerized and deployed on Elastic Beanstalk, its crucial for it remains up no matter what as failing to do so will eventually make the system non-accessible. Second is Cognito because user will only be allowed to access the system if he/she is successfully authenticated otherwise they won’t be granted access as they will not have proper permissions and roles assigned to them which would indicate that no unknown entity is logged into the system. Lastly, Lambda functions performs major backend computations for our system and their unavailability means that system will not be able to process the ticket confirmation for the user.

COMPARISON WITH BENCHMARK

Determine which architecture you've learned about in class your group's project is most like. List it, and then describe any differences you see between your design and the fundamental or advanced architecture from class.

As discussed in the earlier part of the report and the previous report we have listed the requirements and the significance of each service we are using to develop the project. In the earlier part it is mentioned that we will be using architecture of a basic cloud app.

Here we are using an updated form of the architecture listed in the figure below.

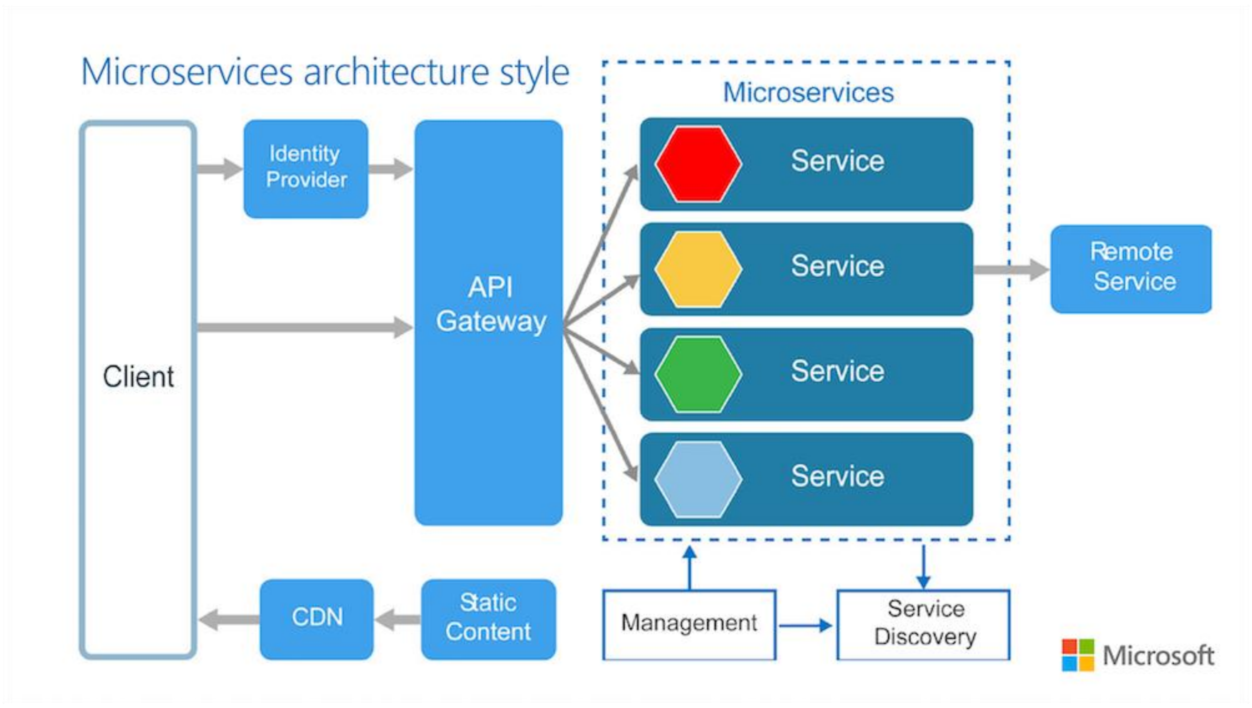


Figure 2 Microservices Architecture by Microsoft

Here we could see that this is the boiler plate kind of thing for the development of architecture, the identification provider and the client relate to the AWScognito service through the API Gateway. Following that we are using a similar kind of approach where every service is triggered through the API gateway and the data is also saved on the cloud.

We updated some things from this approach as there was need of lambda function to be triggered in one use case where we need to implement the conditional statement. Continuing that the development of architecture is done putting the difficulty and the scalability in mind.

According to the architectures which we went through in the class this are similar to them as we are not implementing any extra services and we are going with the similar approach and flow. So the whole architecture could be divided in there parts, Authorization and API Gateway, Serverless services and, database and storage as referred to the next image.

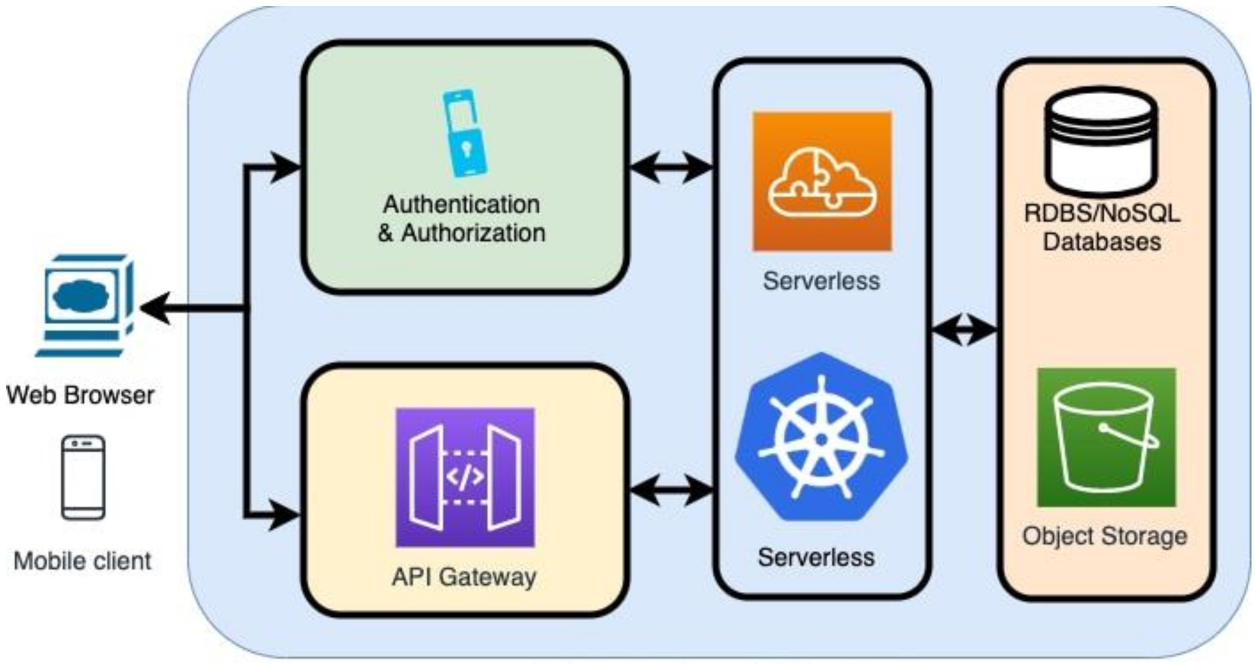


Figure 3 Cloud Based Architecture

This image presents the basic raw idea of all the flow of the system as this is similar to the architecture image and the flow of the data in the whole application is same for the both of them.

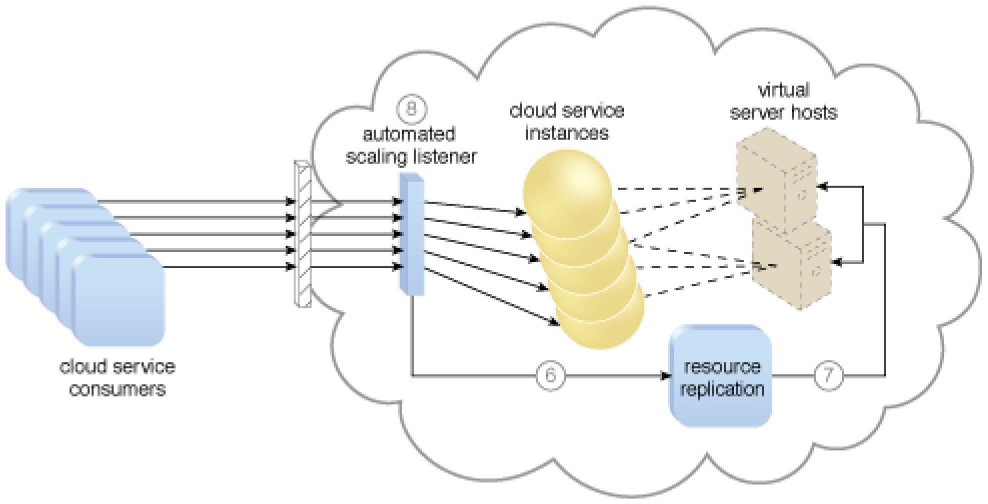
Advantages of this approach:

- This approach is significantly easy compared to other complex architectures
- The data flow in the systems is absolute and having the single database helps to remove the unintended data and resolves the issue of bad read and badly write.
- The whole architecture is based on the AWS services only, so it is easy to implement this compared to multiplatform services.

COMPARISON WITH OTHER ARCHITECTURES

Since the architecture defined by us is a microservice based architecture where microservices drive the entire application. However, this architecture can also be replaced by the following architecture based on the resources.

1) Dynamic Scalability architecture
   This architecture would provide better scalability than the microservice based architecture defined by us. In this architecture the listener can either scale horizontally or vertically or relocate the services. This would be helpful in either handling a larger user base while scaling up or scaling out which is entirely automated.



   However, this architecture would require professionals to setup and maintain the automated scaling listeners whereas incase of microservices once it is setup it requires very low maintenance. This architecture could prove costly compared to the architecture proposed by us as any small mistake would lead to huge bills.

2) Service Load Balancing Architecture

   In this architecture a load balancer performs the scaling of cloud services. It could be used in addition to our architecture but it required replication to ensure nodes work on the same data. Hence we are not using this architecture as replication is a costly process. However, this architecture has higher availability, performance and scalability but is less cost effective than our architecture. Eventhough this architecture has many benefits over our current architecture this is more complex than the proposed architecture which is due to the load balancing.

3) Cloud Balancing Architecture

Though cloud balancing and dynamic scaling architecture are the same as they use automatic scaling listener. IT resources can be load balanced using this architecture across multiple cloud platforms. Though the performance and scalability, reliability, availability the functionality is primarily based on combination of automated scaling listener and failover system mechanisms. This architecture has a complicated implementation and there are no native failure detection or fault tolerance and dynamic load balancing. Since microservices are easy to implement and failure of one microservice doesn’t affect the efficacy of others.

REFERENCES

[1] “AWS Elastic Beanstalk – Deploy Web Applications,” *Amazon Web Services, Inc.*, 2019. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Accessed: 06-Nov-2021]

[2] “Amazon Cognito - Simple and Secure User Sign Up & Sign In | Amazon Web Services (AWS),” *Amazon Web Services, Inc.*, 2021. [Online]. Available: https://aws.amazon.com/cognito/. [Accessed: 07-Nov-2021]

[3] “AWS Secrets Manager | Rotate, Manage, Retrieve Secrets | Amazon Web Services (AWS),” *Amazon Web Services, Inc.*, 2011. [Online]. Available: https://aws.amazon.com/secrets-manager/. [Accessed: 07-Nov-2021]

[4] “Amazon API Gateway | API Management | Amazon Web Services,” *Amazon Web Services, Inc.*, 2021. [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed: 07-Nov-2021]

[5] “What is AWS Lambda? - AWS Lambda,” *Amazon.com*, 2021. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. [Accessed: 07-Nov-2021]

[6] “Amazon Simple Notification Service (SNS) | Messaging Service | AWS,” *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/sns/. [Accessed: 07-Nov-2021]

[7] “Four Architecture Choices for Application Development in the Digital Age,” *Ibm.com*, 06-Jan-2020. [Online]. Available: https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development. [Accessed: 07-Nov-2021]

[8] Chose the right Cloud Application, *Twitter*, 2021. [Online]. Available: https://twitter.com/azure/status/999915523008675841. [Accessed: 07-Nov-2021]