Tool & Application Development Report

CO463 - Cloud Computing

30th May 2020

Team Members

Roll Number	Name	Email ID
16CO104	Ashwin Joisa	ashwinjoisa@gmail.com
16CO124	Mehnaz Yunus	mehnaz138@gmail.com
16CO125	Mishal Shah	shahmishal1998@gmail.com

Contents

Virtual Clinic

Overview of Application

Virtual Clinic is based on the concept of an integrated care system. Stakeholders for the system are Doctors, Patients, Labs, Chemists, and an Administrator. A patient can make a consultation request to the system by entering the symptoms and the system forwards the consultation request to a doctor of concerned specialty. The doctor generates a prescription compromising Diagnosis, Medicines, and Lab Request (if needed). The prescription is received by the patient. The prescription received by the Chemist is to provide medicines to the patient via offline delivery. Labs could also use the same prescription to collect the specimen from the patient and make delivery of lab reports based on lab tests and also update the report on the system to view for the patient anytime. For Chemists and Labs, only need to know the information will be displayed thereby protecting Patient Privacy.

Architecture Diagram

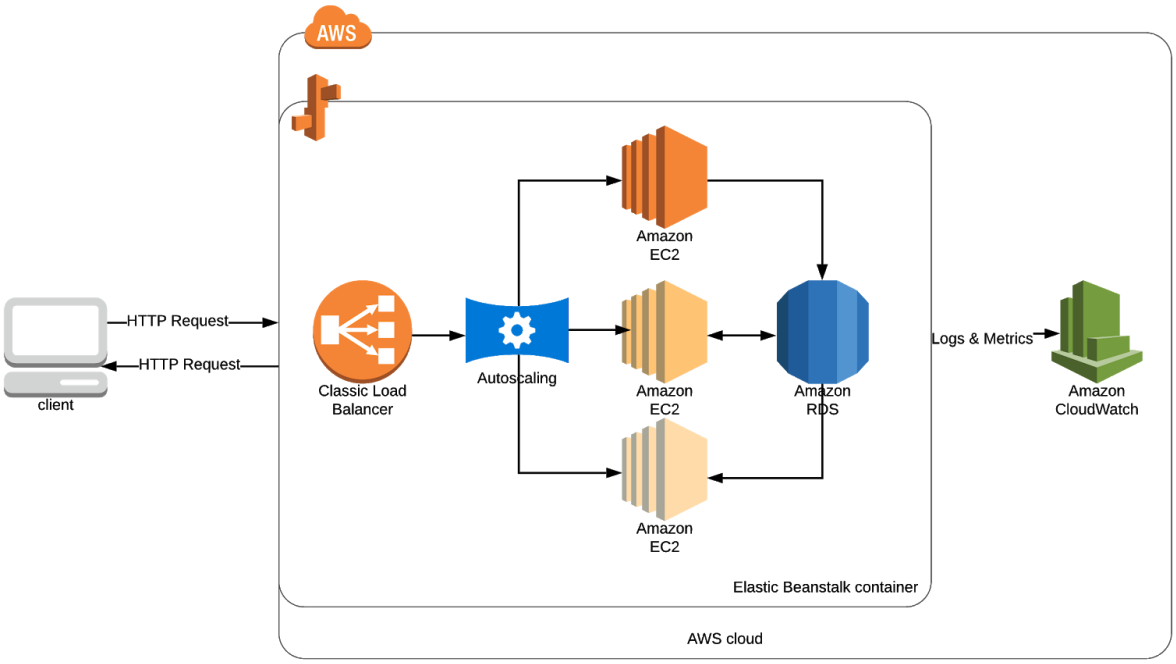

Fig 1.1: Architecture Diagram of the application deployed on AWS

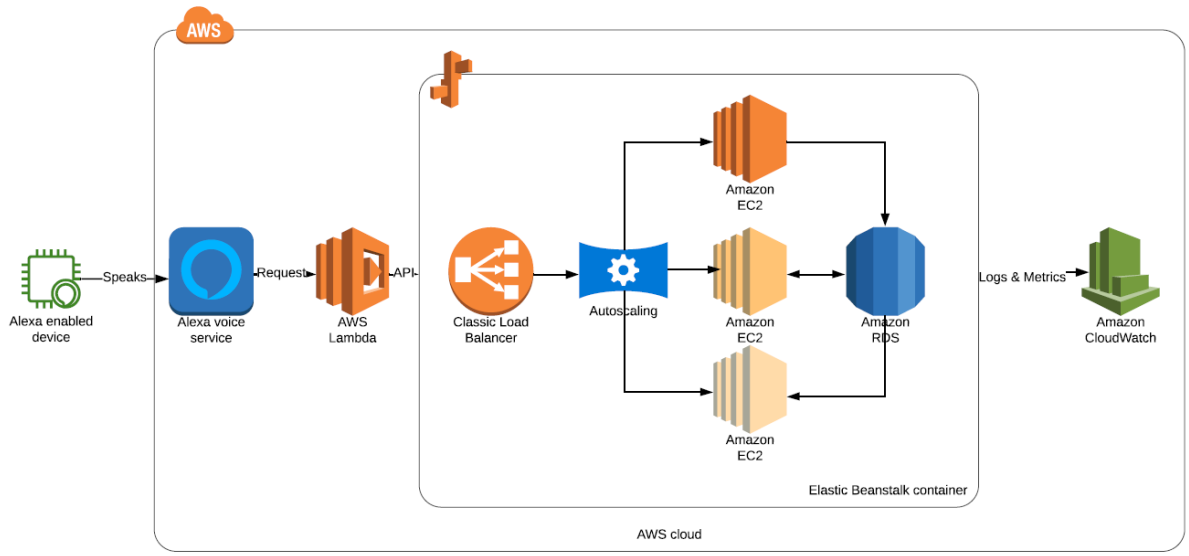

Fig 1.2: Architecture Diagram of the alexa skill deployed on AWS

Cloud Platform: AWS (Amazon Web Services)

Reasons for choosing AWS as Cloud Platform

- Easy to use
- Flexible
- Cost-Effective
- Reliable
- Scalable and high-performance
- Secure
- Supports elastic load balancing

Cloud Features Used

Elastic Beanstalk

AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services. Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring. The user also retains full control over the AWS resources powering the application and can access the underlying resources at any time.

- Cloudwatch

  Amazon CloudWatch is a monitoring service for AWS cloud resources, such as Amazon EC2 instances, Amazon DynamoDB tables, and Amazon RDS DB instances, as well as custom metrics, and AWS applications. It can collect and track metrics and log files, set alarms, and automatically react to changes in AWS resources. It can be used to gain system-wide visibility into resource utilization, application performance, and operational health. These insights can help to react and keep the application running smoothly.

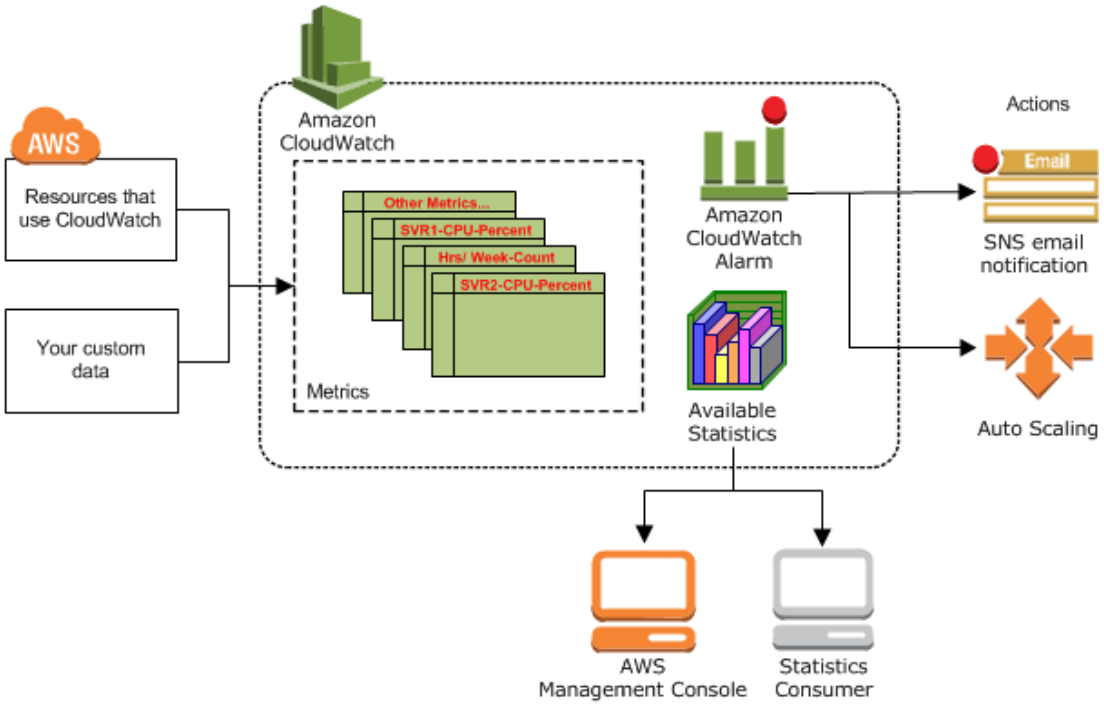

Fig 2: Architecture of Amazon Cloudwatch

- Auto Scaling

  Elastic Beanstalk automatically scales the application up and down based on the application's specific need using easily adjustable Auto Scaling settings. For example, CPU utilization metrics can be used to trigger Auto Scaling actions. With Elastic Beanstalk, the application can handle peaks in workload or traffic while minimizing costs. The EB environment contains an Auto Scaling group that manages the Amazon EC2 instances in the environment. In a single-instance environment, the Auto Scaling group ensures that there is always one instance of running. In a load-balanced environment, the group can be configured with a range of instances to run, and Auto Scaling adds or removes instances as needed, based on load.

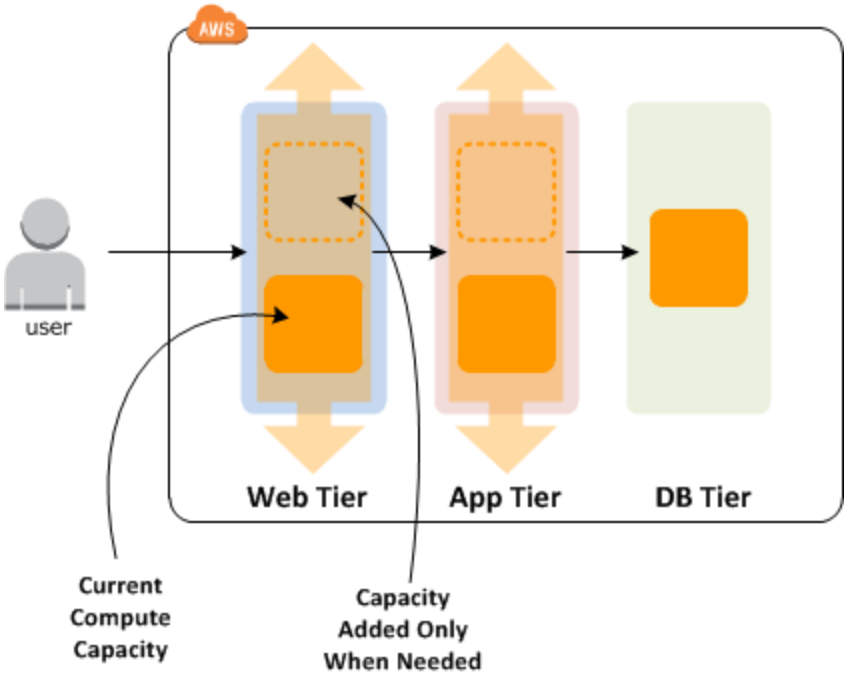

Fig 3: Explanatory diagram on Auto Scaling

- Elastic Load balancing

  When load balancing is enabled, AWS Elastic Beanstalk creates an Elastic Load Balancing load balancer for the environment. The load balancer distributes traffic among the environment's instances.

  Elastic Beanstalk supports these load balancer types:

1. *Classic Load Balancer* – The Elastic Load Balancing a previous-generation load balancer. Routes HTTP, HTTPS, or TCP request traffic to different ports on environment instances.
2. *Application Load Balancer* – An application-layer load balancer. Routes HTTP or HTTPS request traffic to different ports on environment instances based on the request path.
3. *Network Load Balancer* – A network layer load balancer. Routes TCP request traffic to different ports on environment instances. Supports both active and passive health checks.

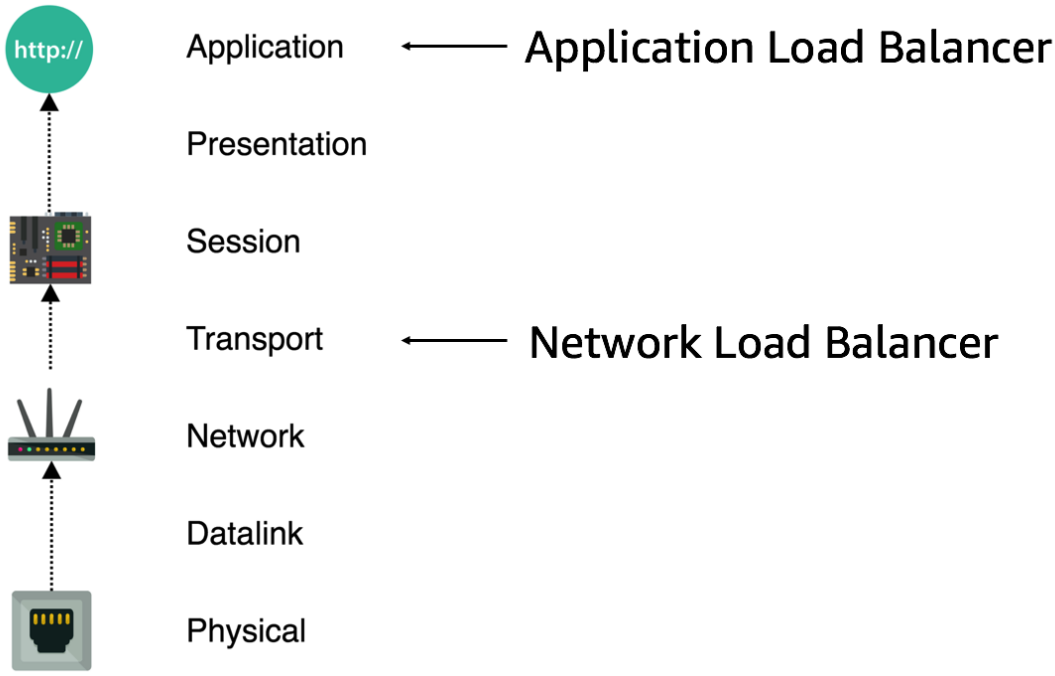

Fig.3: Representation of Load Balancer w.r.t OSI Model

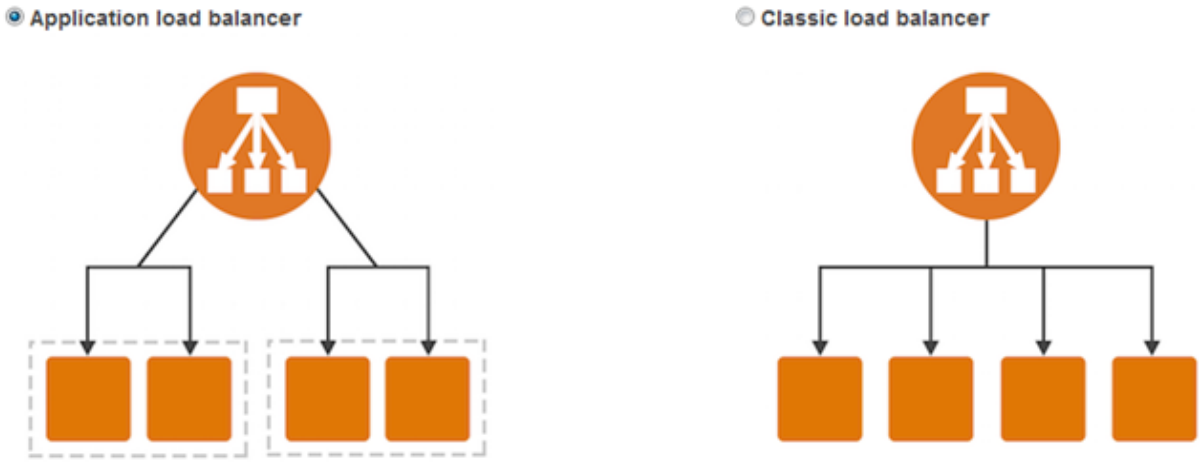

Fig4: Diagramatic representation on the difference in application & classic load balancing

Amazon RDS

Amazon Relational Database Service (RDS) (or Amazon RDS) is a distributed relational database service by AWS. It is a web service running in the cloud designed to simplify the setup, operation, and scaling of a relational database for use in applications. Administrative processes like patching the database software, backing up databases, and enabling point-in-time recovery are managed automatically. Scaling storage and compute resources can be performed by a single API call. It supports various features like Multi-Availability Zone Deployment, read replicas, performance monitoring, backups. It gives us multiple database engines -- Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server.

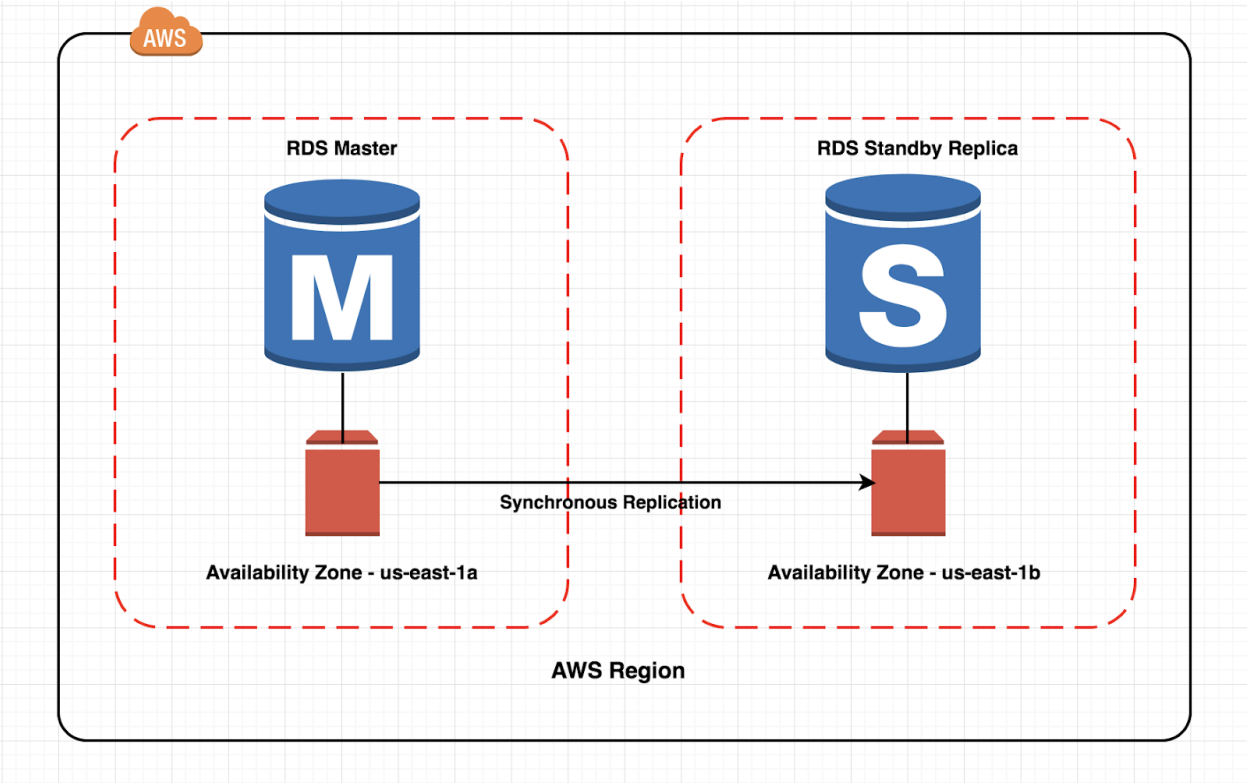

Fig 5: Representation of replication in AWS RDS

Alexa Skills

Alexa is a virtual assistant AI technology developed by Amazon. It is capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audiobooks, and providing weather, traffic, sports, and other real-time information, such as news. Alexa provides a set of built-in capabilities, referred to as skills. A custom interaction skill that can define the requests the skill can handle and the words users say to invoke those requests are made.

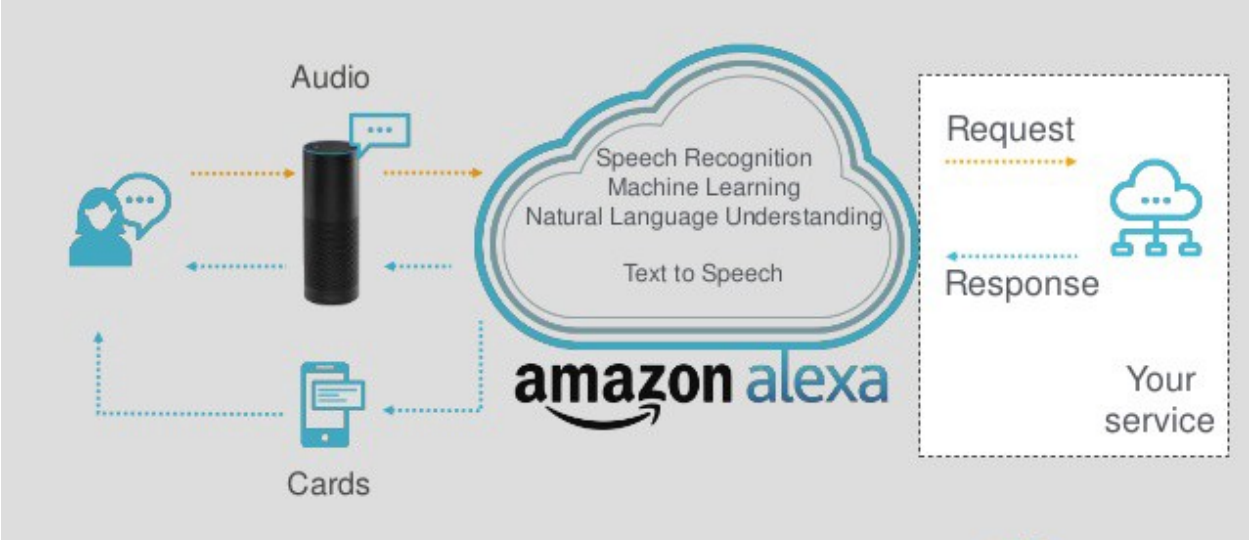

Fig 6: Alexa architecture overview

Lambda (Serverless Architecture)

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code. AWS Lambda lets us run code without provisioning or managing servers, and we only pay for the computing time we consume. It automatically scales the application by running code in response to each trigger.

Results

For our application, the number of **requests to the server on a per-minute basis** was tracked using CloudWatch and this was the criteria used to scale up and down instances/servers.

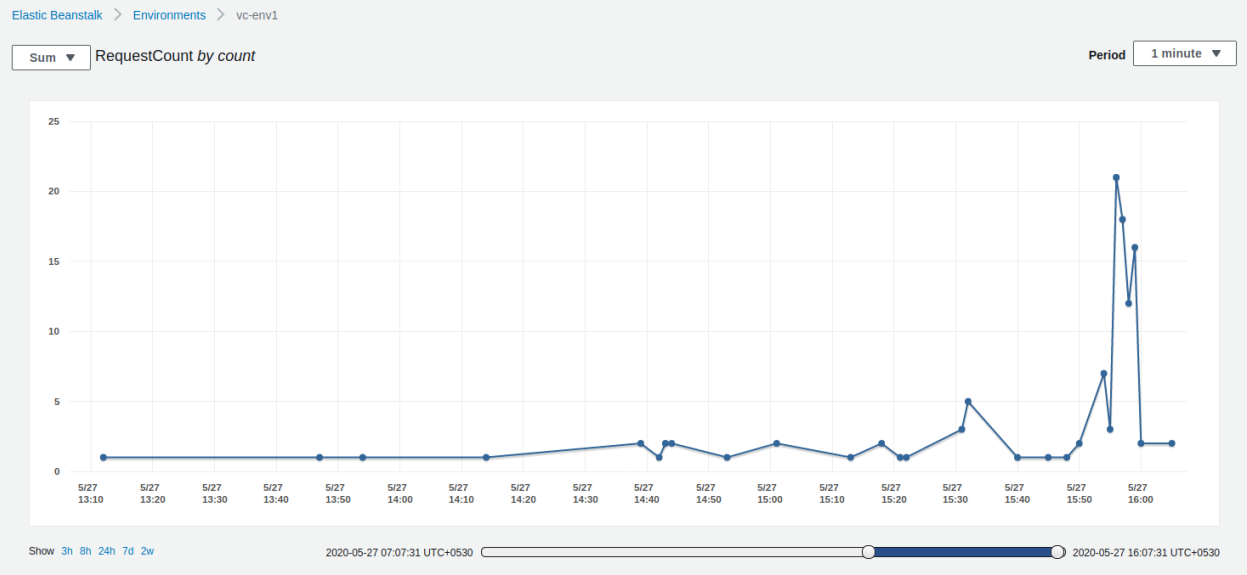

Fig 7: Number of requests to the server on a per-minute basis

Scaling up and down was done using an **Auto-Scaling** group. For the application we deployed, we configured a minimum of 1 instance and a maximum of 3 instances. An event of 5 or more requests to a server per minute would trigger the scale-up by 1 instance and less than 3 would trigger a scale-down of 1 instance. These values were set for demonstrational purposes.

From Fig7, we can see that past 15:55, there was a huge increase in the number of requests, and this led to the spawning of more instances. As we can see in the below image a third instance is being initialized.

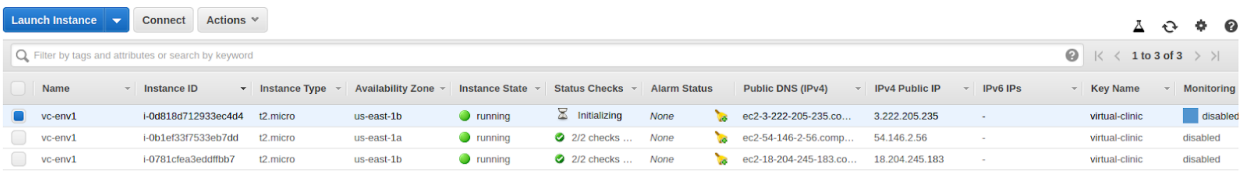

Fig 8: Initializing the 3rd instance by auto-scaling.

Past 16:00, we can see the requests dropping by a large degree. This triggers a scale-down and the auto-scaling group shuts down one of the servers.

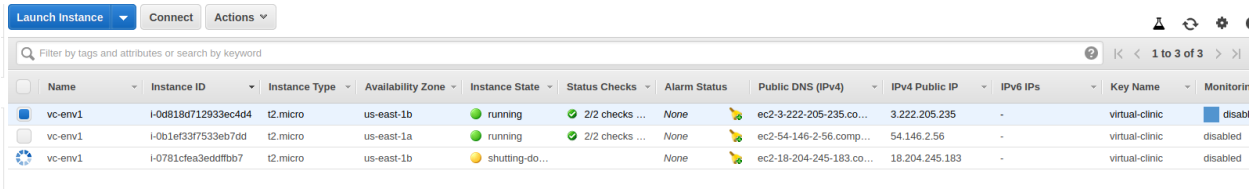

Fig 9: Shutting down the first server.

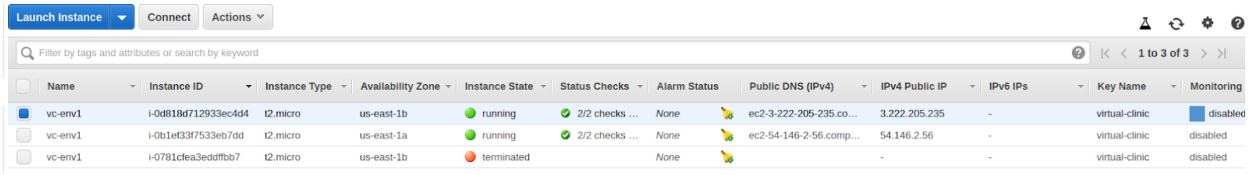

Fig 10: The first server terminated.

Upon a further decline in the number of requests, a second instance was also terminated.

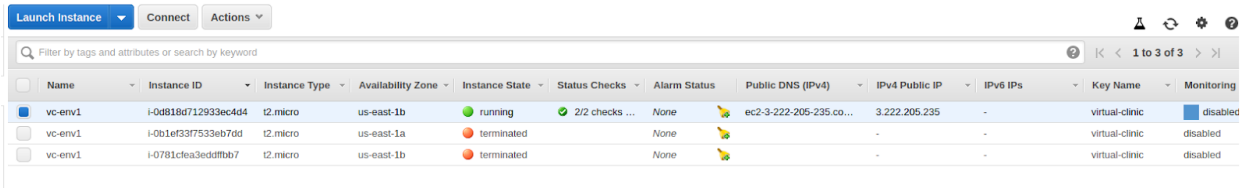

Fig 11: The second server terminated.

The list of events that occur can be tracked in the events page, and as shown below, removal and addition of instances can be seen with the timestamps.

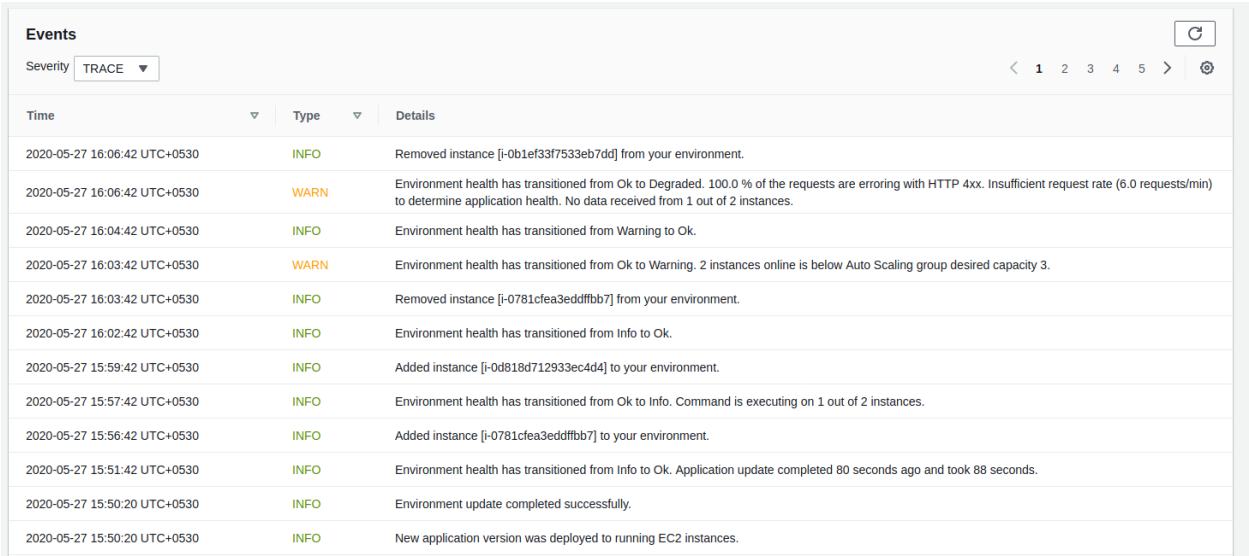

Fig 12: Events

For load balancing, we have deployed a **classic load balancer** that balances load within the auto-scaling group that was created. The configuration has been set to balance the load between the multiple instances based on the server which has the **least load**. Further to test if the load balancer is functioning as expected, we integrated a feature into the application to show the server IP address. Using this we can see that consecutive requests are not redirected to the same server

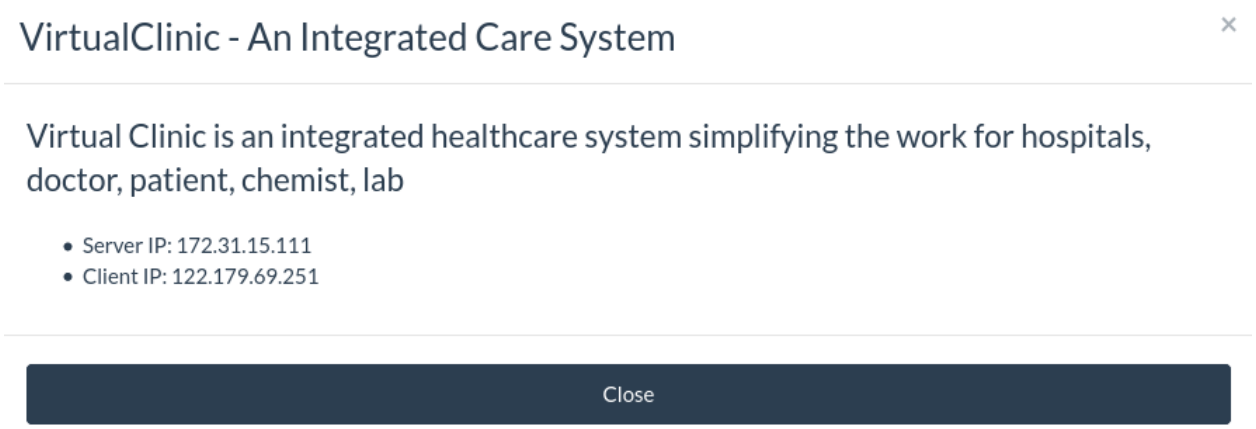

Fig 13: Server IP address is shown in the application.

We also integrated the application with Voice based interface - Alexa to make it easier for doctors to know the upcoming type of appointments and help them know the schedule for the day. This would help them schedule other meetings. Alexa works with the AWS Lambda serverless architecture thus invoking only when the request is triggered. The skill is invoked by saying *"Alexa, Open Virtual Clinic"* and then the necessary intents are set to make the skill work as per the speech.

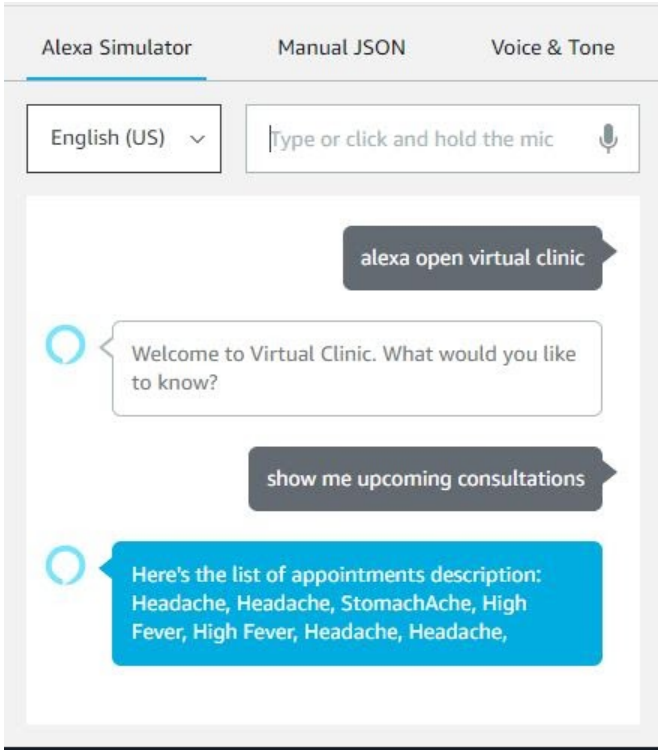

Fig 14: Working of Alexa skill

Conclusion

Cloud platform offers convenient, on-demand network access to a shared pool of configurable computing resources such as servers, storage, applications, and services, that can be rapidly provisioned and released with minimal effort or service provider interaction. With this, one can see huge benefits to deploying applications on the cloud as follows

1. **Easy provisioning of resources**
   The process of provisioning resources is automated and hence is made easy for the user. It is also on an on-demand basis.
2. **Rapid elasticity**
   Cloud computing resources can be provisioned rapidly and elastically. Cloud resources can be rapidly scaled up or down based on demand. This is used to meet the demands of the users of an application, where the number of requests per unit time can vary drastically over time.
3. **Measured service**
   Cloud computing resources are provided to users on a pay-per-use model. The usage of the cloud resources is measured and the user is charged based on some specific metric. This is unlike a conventional model where a fixed number of servers are bought beforehand. It also leads to reduced costs.
4. **Performance**
   Cloud computing provides improved performance for applications since the resources available to the applications can be scaled up or down based on the dynamic application workloads.
5. **Reliability**
   Applications deployed in cloud computing environments generally have higher reliability since the underlying IT infrastructure is professionally managed by the cloud service provider.

References

- https://medium.com/containers-on-aws/using-aws-application-load-balancer-and-network-load-balancer-with-ec2-container-service-d0cb0b1d5ae5
- https://realpython.com/deploying-a-django-app-and-postgresql-to-aws-elastic-beanstalk/
- https://medium.com/tensult/elastic-beanstalk-with-autoscaling-81791a198095
- https://aws.amazon.com/elasticbeanstalk/
- https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/migrate-to-application-load-balancer.html
- https://docs.aws.amazon.com/