MyStartup’s AWS Cloud Architecture Design document

Customer: MyStartup, Site ID: ABC Document, version: 1.0v

Contents

[Abstract 1](#_Toc511396063)

[Introduction to AWS 1](#_Toc511396064)

[Proposed architecture will be as below 2](#_Toc511396065)

[Design Advantages and key points 3](#_Toc511396066)

[Cost Estimates 4](#_Toc511396067)

[Tools used 5](#_Toc511396068)

[References 5](#_Toc511396069)

# Abstract

This document describes the proposed AWS Architecture for hosting LAMP stack along with REST API portal which enables clients to login and do the tasks. The scope of the document is to determine right architecture and addresses the requirements listed.

# Introduction to AWS

Amazon Web Services is a secure cloud service platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. There are several regions and availability zones across the globe.

Operational benefits

1. Creating a culture of cost management
2. Easy to use
3. Flexible and cost effective
4. Secure, Scalable, high performing and Reliable.

For more reading – Kindly refer to [References section](#_References)

AWS Cloud environment can be accessed by – AWS Management console, CLI, SDK toolkit, API. Based on the user and nature of job appropriate access mode can be used.

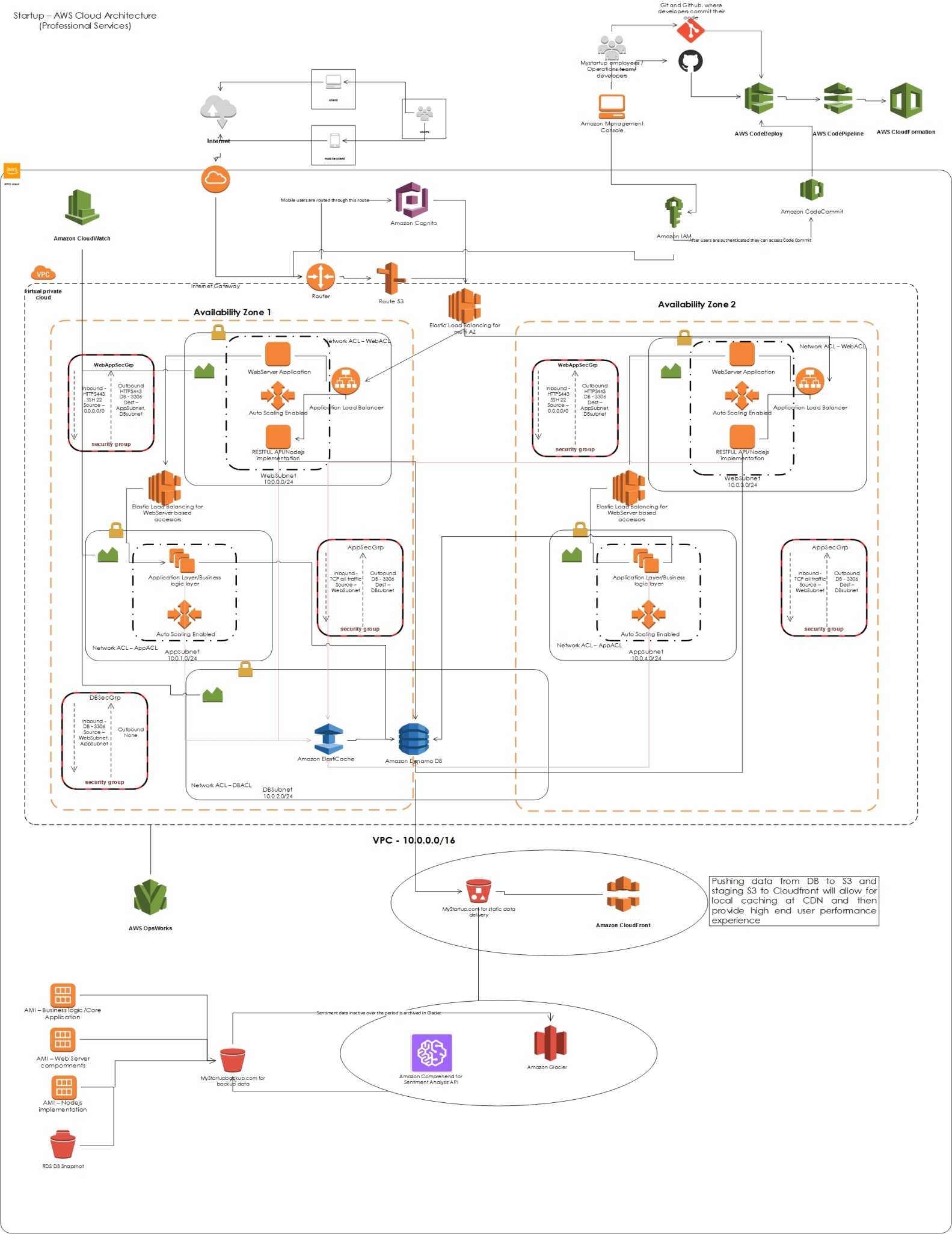
Current Problem Definition - <Referring to the document (pdf) provided.

We can approach this problem in various ways, however below proposed architecture suits well in terms of manageability, performance, elasticity, security, fault tolerant and recoverable.

However while architecting there were few assumptions made, and they are listed below

* MyStartup.com is the company name.
* This stack has 3 layers – Web/Internet interface layer (hosting Apache and acts as Web Server) there is Nodejs implementation for queries coming in from mobile app. (As mobile app already has logic, there is no need for us to connect it with AppLayer, hence it directly its DBlayer, Application Layer (where core components of Business logic is residing) and Database Layer.
* There is no cross region deployment of this stack, if there is a plan for cross region deployment then Global Dynamo DB tables can be used.
* Active – Active architecture.
* Dynamo DB tables are periodically backed-up.
* There are certain services which are not highlighted ex. SNS, Auto Scaling or terminate instances when particular threshold is crossed, For Continuous integration, no tools are considered – Jenkins, CDN, edge location etc.
* Ports to be used between WebLayer and AppLayer. Allowing all traffic at present between these 2 layers. Ports will be set based on business logic and services running on Application.
* Between WebApp and DB layer (direct), if many mobile users are connecting then we can enable auto scaling on Dynamo DB which is inbuilt feature.
* Number of users steadily increases over the period of time.

# Proposed architecture will be as below



Introduction to various services used in the architecture and their flow.

1. Web browser, mobile application user will connect to [www.mystartup.com](http://www.mystartup.com).
2. For auto-scaling, we need AMI to be defined and stored. Also any backups which are needed are to be stored in S3. So that they are globally available (internal network) and it is cheaper as well. We can set policy on S3 to further reduce costs by archiving objects to Glacier.
3. In order to capture sentiment analysis of users, we can make use of Amazon Comprehend. Once we have arrived at conclusion, we can further store the data in Backup bucket and archive it.
4. In order to automate the deployment we can define the infrastructure in code and save it in. We can spin the entire architecture in few clicks by using Cloud Formation service. This can be an endpoint (last candidate) service in cloud automation step/process.
5. Business which is looking for CI/CD then, developers can code the logic/program/app and then use CodeCommit as VCS. CodeDeploy will pick this up and build it with the build engine selected. CodePipeline can be used to automate this entire step by step process.
6. AWS provides monitoring, alerting, events options. Use case based dashboards, alert definitions and actions can be written.

Note – Above mentioned AWS services are brief introduction to their job in current architecture. However, if there is further reading required. Kindly visit <https://aws.amazon.com/documentation/?nc2=h_ql_d&awsm=ql-5>

# Design Advantages and key points

1. Architecture is highly scalable. There is space to scale it horizontally at each layer and at spread it across more Availability zones/Regions too. This is an active-active site hence more traffic handling.
2. Internally resources are scalable. Use of Auto Scaling service will enable to scale out and scale in instances. S3 buckets are highly scalable with respect to storage option. LB will enable routing of traffic to underneath instances in a smooth way so that they are not overloaded. Dynamo DB provides additional control on read/write IOPS.
3. As this is a startup, we can begin without throttling of API, however once we see increase hit on the server then we can introduce Amazon API Gateway. Which will work in below fashion

* Receive incoming request
* Check throttling config, and if it is above allowed rate.
* Return “429”

Mobile users / Web site users and other services

Execute Backend call

1. RDS DB could have been used for this case, however if we want to improve the performance further we will pick to go with Dynamo DB, which, will give deeper control on write/read IOPS and is much cheaper than RDS DB. Along with this we are implementing ElastiCache which will further reduce load on DB.
2. When there are users who are connecting from different region, and in order to reduce latency we are making use of Cloud Front – CDN. This will cache contents in nearest location to the user and provide it without delay. However, very first request might be of few seconds delay.
3. In order to provide secure connectivity for users connecting from social media and custom sites, we can make use of AWS Cognito. This will provide users with the ability to sign up directly from social media sites and give seamless access along with synchronized user preferences.
4. AWS OpsWork can define the architecture of the application in terms of layers i.e. App layer, DB layer and Web layer and the specification of each component – package installation, software configuration This will provide self healing at the application level. If self healing is required at infrastructure level then we have constant monitoring which can trigger Auto Scaling to bring in new instances using the AMIs, LBs to remove faulty instances from traffic. CloudFormation which can handle spinning up of updated versions after they are tested using CodePiples and then delivered to live.
5. Collected data need sentiment analysis we can make use of Amazon Comprehend, which will enable us to analyze user sentiments and then store if for future use into S3 and then Glacier.
6. S3 will provide security for data, It supports AES256 and AWS-KMS. By default, they will be disabled. In our case we will need to enable them.
7. There is IAM which is a must for every AWS account. We can create users, policy, group etc in it. It can also be integrated with corporate identity stores which give flexibility to handle users more easily. We can create required groups and add users based on their job roles. These groups can be defined with policy of access to underlying resources that will make AWS more secure, further securing can be done using MFA (in order to avoid human errors). Cloudtrail can be enabled for auditing purpose.
8. There is continuous need to improvise and deliver, and we need developers to be on top of it to build/develop solutions. Defining pipelines and routing it to build the environment by itself will speed up things. This can be done by using CodeCommit, CodeDeploy, CodePipeline and then hand it over to CloudFormation for actual deploy in test/prod. Codecommit will help in VCS, CodeDeploy will help in building, testing. For deployment we can make use of CloudFormation, ECS, Beanstalk based on use cases.
9. Once we have these templates created, CloudFormation will assist in rapidly building the environment. These templates are easy to manage, replicate the infrastructure and upgrade.
10. ELBs perform health checks of instances and replaces faulty instances with new one.
11. Auto scaling is enabled with launch configuration so that there, there is constant check on health metrics and scale out when needed and scale down when load is reduced. This will be kind on pocket and dynamic scaling will help in peak business hours.
12. All these above resources allows us to scale horizontally at “Web Server”, “Application/Business logic” and “Database” level. If we want it to be furthermore scalable we can decouple few modules.

# Cost Estimates

* This is an estimate only, this can be different as compared when it is actually implemented.
* Calculator shows it might cost around $1700 per month and $4500 for upfront payments of EC2 instances. Making upfront payment for EC2, will have slight discount on the price.
* Based on this price, it is 0.68% of the fund allocated in total. Which definitely is a very cheap subscription per month.
* This stack comes with - self healing, auto scaling, fault tolerant, high performing. So that you organization can concentrate on business logic and market study.

# Tools used

1. Microsoft suite – Word document, Visio 2003.
2. Snipping Tool
3. Simple Icons from AWS site - <https://aws.amazon.com/architecture/icons/>
4. <https://calculator.s3.amazonaws.com/index.html> for rough estimates of prices.

# References

* Amazon documents section.
* Amazon videos on YouTube.
* <https://aws.amazon.com/blogs/aws/aws-mobile-services/>
* <https://aws.amazon.com/blogs/publicsector/achieve-total-cost-of-operation-benefits-using-cloud/>
* <https://d0.awsstatic.com/whitepapers/total-cost-of-operation-benefits-using-aws.pdf>
* <https://aws.amazon.com/documentation/?nc2=h_ql_d&awsm=ql-5>
* <https://calculator.s3.amazonaws.com/index.html>