Cloud Architecture

Team 35

Aswin Itha Smayana Pidugu Vamsi Madhur Varada
{aitha, spidugu, vvarada3}@ncsu.edu

"We, the team members, understand that copying & pasting material from any source in our project is an allowed practice; we understand that not properly quoting the source constitutes plagiarism.

All team members attest that we have properly quoted the sources in every sentence or paragraph we have copy & pasted in our report. We further attest that we did not change words to make copy & pasted material appear as our work."

Contents

1 Introduction

1.1 Motivation

To gain knowledge and skills in cloud technology, across a wide range of cloud providers. To demonstrate a strong understanding of cloud services by designing a cost, performance optimized solutions.

1.2 Executive Summary

This document provides valuable information on design principles, best practices, and tradeoffs that are made while developing a system.

2 Problem description

2.1 The Problem

Voting App

The Voting App is a simple application built using Python, Node.js, Redis, NoSQL, and .NET. It allows you to vote for your favorite pet - dog or cat. The percentage of votes is posted as results on a different page. The application is taken from the docker samples developed and maintained by Docker Hub.

The source code can be found at:
https://github.com/dockersamples/example-voting-app.git

Voting-App aims to provide uninterrupted and best service for the participants. As cloud architects, it is our responsibility to make this dream a reality by utilizing the Cloud Services available. We will evaluate and document the essentials to move our application to cloud space.

2.2 Business Requirements

BR 1 - Provide Operational Excellence [2]
: Help in running workloads effectively, gain insight into operations, and continuously improve supporting processes and procedures to deliver business value.

BR 2 - Deliver Elevated Security [2]
: Protect data, systems, and assets in a way that can improve the security posture.

BR 3 - Maintain High-Reliability [2]
: Ability of a workload to perform its intended function correctly and consistently

when it's expected to. Ability to operate and test the workload through its total lifecycle.

BR 4 - Efficient Performance [2]
Efficiently use computing resources, and maintain effectiveness as demand changes and technology advances.

BR 5 - Achieve Low-Cost [2]
Help build and operate cost-aware workloads that achieve business outcomes while minimizing costs and maximizing its return on investment.

BR 6 - Increased Sustainability [2]
Understand the impacts of the services used, quantify impacts through the entire workload lifecycle, and apply design principles and best practices to reduce these impacts.

2.2.1 BR and TR Relation Table

BR1	TR 1, TR 2, TR 3, TR 4, TR 5, TR 6, TR 14, TR 22, TR 23
BR2	TR 1, TR 6, TR 7, TR 8, TR 9, TR 10, TR 11, TR 19, TR 26
BR3	TR 3, TR 4, TR 5, TR 12, TR 13, TR 14, TR 15, TR 18, TR 20, TR 21, TR 22, TR 23, TR 24
BR4	TR 14, TR 20, TR 21
BR5	TR 2, TR 4, TR 6, TR 8, TR 12, TR 15, TR 16, TR 25, TR 26
BR6	TR 17

2.3 Technical Requirements

TR 1 - Provide Monitoring
Related BRs: BR 1 - Provide Operational Excellence, BR 2 - Maintain High-Reliability
Justification for BR 1 - Provide Operational Excellence, BR 2 - Maintain High-Reliability

- Measuring metrics like CPU utilization, response time, workloads, etc would help understand the health of the application and continuously improve the system to deliver business value. This leads to having a reliable and workload-efficient system.

TR 2 - Provide Version Control and Deployment Management [3][4][5]
Related BRs: BR 1 - Provide Operational Excellence, BR 5 - Achieve Low-Cost
Justification for BR 1 - Provide Operational Excellence:

- Version control helps to keep track of the deployment process and the changes being made to the application. In case of a disaster, it helps to roll back changes and thus improves operational excellence.
- The use of standard deployment pipelines also reduces the errors caused by manual deployment and smoothens the process of building, testing, and deploying the application helping in achieving operational excellence.

Justification for BR 5 - Achieve Low Cost:

- Deployment management reduces the time and effort required by the development team on the operations reducing the cost of software development.

TR 3 - Disaster Management and Recovery [6][7][8]

Related BRs: BR 1 - Provide Operational Excellence, BR 3 - Maintain High-Reliability

Justification for BR 1 - Provide Operational Excellence, BR 3 - Maintain High Reliability:

- Any system is prone to disaster and its occurrence is very tough to predict. So, defining disaster failure procedures like Recovery Time Objective(RTO), Recovery Point Objective(RPO), and DR strategy helps in getting the system back to a good state thus increasing reliability and business value.
- Providing standard and best practices support like periodic automated testing of a disaster recovery plan helps team understand the recovery procedures and find areas to be improved.

TR 4 - Provide workload and key performance indicators [9][10]

Related BRs: BR 1 - Provide Operational Excellence, BR 3 - Maintain High Reliability, and BR 5 - Achieve Low-Cost

Justification for BR 1 - Provide Operational Excellence, BR 3 - Maintain High Reliability:

- Defining the workloads and KPIs help in testing and evaluating the health and success of workloads thus increasing operational excellence and reliability.

Justification for BR 5 - Achieve Low Cost:

- The indicators also help in analyzing and understanding the business thereby reducing investments in non-performing areas and saving money.

TR 5 - Provide Process Alerts [11][12]

Related BRs: BR 1 - Provide Operational Excellence, BR 3 - Maintain High-Reliability

Justification for BR 1 - Provide Operational Excellence, BR 3 - Maintain High Reliability:

- By setting up alerts, disasters can be prevented at an early stage as the data can be studied and patterns can be recognized. The alerts help in identifying infrequent problems and give special attention at the time of occurrence. This helps in increased operational excellence, and reliability.
- Defining push notifications also helps the customers understand and be informed about the impact of a problem. This provision helps in improving operational efficiency with the end users.

TR 6 - Provide Account Management [13][14]

Related BRs: BR 1 - Provide Operational Excellence, BR 2 - Deliver Elevated Security BR 5 - Achieve Low-Cost

Justification for BR 2 - Deliver Elevated Security:

- With account management, different users can be divided into different groups and the permissions to each account can be easily handled centrally increasing the security of the application and ease of management.

Justification for BR 1 - Provide Operational Excellence:

- With account management, the dev, test, and production environments are isolated thus reducing the cascading effect in case of failure and improving operational excellence.

Justification for BR 5 - Achieve Low-Cost:

- By implementing a structure of accounts, costs can be managed throughout the organization.

TR 7 - Implement Identity Management [15][16]

Related BRs: BR 2 - Deliver Elevated Security

Justification for BR 2 - Deliver Elevated Security:

- Identity management helps in defining policies like multi-factor authentication, and periodic password change that helps in allowing only users with permissions to enter the system thus securing it.
- It also defines principles to secure and store secrets of other 3rd party applications, resource sharing, etc.

TR 8 - Provide Application and Resource Logging [17][18]

Related BRs: BR 2 - Deliver Elevated Security, and BR 5 - Achieve Low-Cost

Justification for BR 2 - Deliver Elevated Security:

- This helps in automating responses to events and identifying security events. This helps in preventing and debugging a security incident.

Justification for BR 5 - Achieve Low Cost:

- Analyzing logs saves time for the dev team in troubleshooting and reducing costs. It helps in finding which parts of the application are costlier and try in reducing them.

TR 9 - Provide Infrastructure Protection [19][20][21]

Related BRs: BR 2 - Deliver Elevated Security

Justification for BR 2 - Deliver Elevated Security:

- Implementing the protection schemes in just the application is not sufficient. An attack can be made even on the infrastructure and so it is important to secure it.
- By creating network layers and controlling traffic in them, protection can be achieved.
- Performing vulnerability testing centrally in every deployment helps identify security issues at an early stage.

TR 10 - Implement Data Protection [22][23]

Related BRs: BR 2 - Deliver Elevated Security

Justification for BR 2 - Deliver Elevated Security:

- Using Data Protection controls and defining the life cycle of data helps in securing the data and increasing the security.
- Terminating data that is irrelevant over time also helps in reducing the load on the data servers and in maintaining the existing data.
- Using Data Protection managers, we can limit the availability of the data to the different security groups of people. This increases the protection of the data as not everyone has access to all the data.

TR 11 - Provide Incident Recovery [24][25]

Related BRs: BR 2 - Deliver Elevated Security

Justification for BR 2 - Deliver Elevated Security:

- Identifying the persona responsible for identifying and responding to an incident helps in reducing the time needed for troubleshooting.
- Implementing mechanisms for fallback in case of an incident helps minimize the damage and thus restore security.

TR 12 - Automate and maintain Cloud Quota [26][27]

Related BRs: BR 3 - Maintain High Reliability, and BR 5 - Achieve Low-Cost

Justification for BR 3 - Maintain High Reliability, and BR 5 - Achieve Low Cost:

- Automating requests to increase or decrease the quota when thresholds are reached increases the reliability of the application.
- In case of quota thresholds reaching multiple times, procedures defined to re-calculate the quota and increase it would ensure the prevention of failovers.

TR 13 - Provide high network availability with 5 9's [28]

Related BRs: BR 3 - Maintain High-Reliability

Justification for BR 3 - Maintain High Reliability:

- This means the application is fully operational 99.999% of the time. This implies an average of 6 minutes of downtime per year.
- Implementing high network availability ensures that all the users can access the application at all times thus increasing reliability.

TR 14 - Implement Scalable and Distributed Architecture [29][30][31][32][33]

Related BRs: BR 1 - Provide Operational Excellence, BR 3 - Maintain High Reliability, BR 4 - Efficient Performance

Justification for BR 3 - Maintain High Reliability:

- Implementing an architecture that allows easy scalability is important and makes the system highly reliable.
- To have high scalability of the architecture, loosely couple the dependencies. This is achieved by the use of an Elastic Load Balancer (ELB) or any other load balancer.
- By implementing microservices instead of monolithic services, the scalability of the architecture can be amplified.

Justification for BR 1 - Provide Operational Excellence:

- This also helps in troubleshooting as it is easy to fix small modules than monolithic ones. This makes the operations easy.

Justification for BR 4 - Efficient Performance

- A load balancer handles the varying load of your application traffic across multiple Availability Zones. Load balancers feature the high availability, automatic scaling, and robust security necessary to make the application fault tolerant. To increase the effect of load balancers, scalable architecture is required.

TR 15 - Implement Container Orchestration [34][35]

Related BRs: BR 3 - Maintain High Reliability, and BR 5 - Achieve Low-Cost

Justification for BR 3 - Maintain High Reliability:

- Container Orchestration helps in health-checking existing nodes and spawning new nodes if the response time of an application crosses a set threshold value.
- It aids in handling time-varying workloads by increasing the number of containers and load-balancing (auto-scaling) the requests thus improving reliability.

Justification for BR 5 - Achieve Low-Cost:

- Containers are open source and have very low infrastructure costs. Using inbuilt services of cloud providers (like that of EC2 and EKS of AWS) the cost of obtaining a license is removed and thus reduces the overall costs.

TR 16 - Choose cost-effective tools and services [36]

Related BRs: BR 5 - Achieve Low-Cost

Justification for BR 5 - Achieve Low Cost:

- Choosing cost-effective tools and minimizing the number of tools used at the start of cloud implementation reduce maintenance costs which are critical for cloud operations.
- Analyzing and evaluating the open source software and the ones provided by the cloud providers help in choosing them as these are generally free of cost.

TR 17 - Choose sustainable data centers [37]

Related BRs: BR 6 - Increased Sustainability

Justification for BR 6 - Increased Sustainability:

- Choosing data centers that use renewable energy helps in achieving sustainability goals.

TR 18 - Provide High throughput [38][39][40][41]

Related BRs: BR 3 - Maintain High Reliability

Justification for BR 3 - Maintain High Reliability:

- High throughput enables the system to serve multiple requests simultaneously. This increases the reliability of the system and is achieved by using the best practices.
- By deploying the workload to multiple locations, we avoid a single point of failure.
- By implementing loosely coupled architecture, we avoid a single point of failure and thus increase throughput.

- By throttling requests in case of increased demand for service helps us honor a few requests successfully while slowing the rate of requests or limiting them thus not impacting everyone at the same time.
- Defining the retry policy like exponential backoff helps in controlling and limiting the retry calls thus reducing load on servers at the time of failure.

TR 19 - Perform vulnerability management [42]

Related BRs: BR 2 - Deliver Elevated Security

Justification for BR 2 - Deliver Elevated Security:

- Frequently scan and patch for vulnerabilities in the code, dependencies, and infrastructure to help protect against new threats.
- Use tools to evaluate the security of the application regularly and expose any potential threats.

TR 20 - Low Response time with an average of 300ms [43][44]

Related BRs: BR 3 - Maintain High-Reliability BR 4 - Efficient Performance

Justification for BR 3 - Maintain High-Reliability and BR 4 - Efficient Performance:

- One of the reasons why the response time of an application goes down is when there are a higher number of users than what a server can handle. This can be avoided by requesting more resources to be allocated. This process needs to be automated to avoid any delays.
- By using auto-scaling services and in-memory DBs we can reduce the response time of the application.
- By collecting, and analyzing the performance metrics, a baseline can be established and workload behavior can be analyzed to detect abnormal patterns which help in maintaining low response time.

TR 21 - Provide expandable network topology [45][46]

Related BRs: BR 3 - Maintain High Reliability, BR 4 - Efficient Performance

Justification for BR 3 - Maintain High Reliability and BR 4 - Efficient Performance:

- The network topology helps in ensuring better connectivity and availability of services when using a multi-cloud environment. This increases the reliability and performance of the application.
- The policies defined here make sure that there is no IP overlapping between services when multi-cloud is involved thus increasing reliability.
- By selecting and configuring appropriately sized network solutions will increase the reliability of the workload and maximize performance opportun-

ities.

TR 22 - Implement Change Management [47][48][49]

Related BRs: BR 1 - Provide Operational Excellence, BR 3 - Maintain High-Reliability

Justification for BR 1 - Provide Operational Excellence and BR 3 - Maintain High Reliability:

- Change Management helps anticipate the change in workloads with which the system can be scaled to ensure high reliability and availability.
- By calculating metrics and logging workloads, the change in demand is anticipated and helps in scaling the system.
- Use playbooks to implement change management as it helps in easy automation, avoiding errors and enables easy debugging.

TR 23 - Use Immutable Infrastructure [50][51]

Related BRs: BR 1 - Provide Operational Excellence, BR 3 - Maintain High-Reliability

Justification for BR 1 - Provide Operational Excellence and BR 3 - Maintain High Reliability:

- Using immutable infrastructure using change management helps in rolling back and handling the deployment issues minimizing the effect on the end users thereby increasing reliability and improving operational excellence.
- Immutable Infrastructure enables the use of deployment strategies that help in slow rolling updates while still having the application highly available.

TR 24 - Data Backup and Recovery [52]

Related BRs: BR 3 - Maintain High Reliability

Justification for BR 3 - Maintain High Reliability:

- In case of disaster recovery, the system should be able to retrieve the data securely back to the last good state. This is in line with high reliability.
- While implementing backup, identify data from all the sources, classify the data to backup only the required data
- Define recovery mechanism for the data that is not backed up.

TR 25 - Implement Cloud Financial Management [53][54][55]

Related BRs: BR 5 - Achieve Low-Cost

Justification for BR 5 - Achieve Low Cost:

- Financial Management helps in optimizing the cost by understanding the technology requirements, monitoring existing processes, and keeping up to date with new technologies.

- Increase awareness among the stakeholders by transparently informing them about the costs, sharing success stories, and training

TR 26 - Tenant Identification and Isolation [56]

Related BRs: BR 2 - Deliver Elevated Security, BR 5 - Achieve Low-Cost

Justification for BR 2 - Deliver Elevated Security:

- When there are multiple tenants, tenant identification and isolation become important to secure the tenants. Incoming traffic might contain sensitive information which should not be shared with other tenants residing on the same server. So, tenant identification and isolation are mandatory to secure the users.

Justification for BR 5 - Achieve Low-Cost

- With Tenant identification, the cloud provider would be able to find the traffic coming from each tenant and that helps in generating the right price based on the usage for the consumer.

2.4 TradeOffs

1. **TR 18 - Provide High throughput vs TR 20 - Reduced Response time** [57]

   When you attempt to balance the ongoing need for high request throughput with an low response time for a request, such as counting the number of votes in a particular criteria to declare the results for that criteria, the trade-off between throughput and response time becomes clear. The impact this query can have on request throughput increases as more resources are used to the request, reducing the amount of time you have left to handle other requests. In contrast, the request will take longer the less resources you permit.

2. **TR 23 - Use Immutable Infrastructure vs TR 24 - Data Backup and Recovery** [58]

   Infrastructure that never changes is implied by the term "Immutable Infrastructure." An infrastructure component is never touched again after it has been deployed. If an upgrade, modification, or new deployment is necessary, the old component is destroyed and swapped out for a new one. What if this program was writing to its local disk and it contained information that was important to the program? Here, starting a new machine and destroying the old machine, along with all of its data and disk wouldn't work as we will loose the data.

3. **TR 14 - Implement Scalable and Distributed Architecture vs TR 16 -**

Choose cost-effective tools and services

To have a distributed and scalable system, we could either scale up by running the app on a more powerful VM or reconfigure the system to scale out and run on multiple instances of web server to increase capacity. Both of these solutions require lot of effort and resources which increases the cost of the system.

3 Provider Selection

3.1 Criteria for choosing a provider

We should choose a cloud provider that supports most, if not all of the below requirements.

Tenant Identification TR

1. TR 26 - Tenant Identification and Isolation

Monitoring TR

2. TR 01 - Provide Monitoring

Reliability TRs

3. TR 13 - Provide high network availability with 5 9's
4. TR 24 - Data Backup and Recovery

Performance TRs

5. TR 20 - Low Response time with an average of 300ms
6. TR 18 - Provide High throughput with 1000 to 2000 requests per second.

Security TR

7. TR 07 - Implement Identity Management

Operational Excellence TR

8. TR 05 - Provide Process Alerts

Cost TR

9. TR 16 - Choose cost-effective tools and services

Other TRs

10. TR 14 - Implement Scalable and Distributed Architecture
11. TR 15 - Implement Container Orchestration
12. TR 23 - Use Immutable Infrastructure

3.2 Provider Comparison

We chose the top three cloud providers for comparison.

- Amazon Web Services - AWS
- Microsoft Azure - Azure
- Google Cloud Provider - GCP

Below is a comparison table of providers based on the criteria

TR	AWS	Azure	GCP	Ranking
TR 23	AWS CodeDeploy	Azure DevOps	Cloud Run	Cloud Run is very efficient[59]
TR 20	MemoryDB	Azure Cache	Memory Store	AWS is the least expensive amongst all[60]
TR 15	Elastic Kubernetes Service	Azure Kubernetes Service	Google Kubernetes Engine	AWS has less pre-configuration and therefore more control [61]
TR 26	AWS IAM	Azure Active Directory IAM	GCP IAM	AWS IAM is robust and serverless [62]
TR 24	S3; DynamoDB	Azure Blob Storage	Google Cloud Storage	All are similar[63]
TR 18	MemoryDB	Azure Cache	Memory Store	AWS is the least expensive amongst all [60]
TR 01	AWS CloudWatch	Azure Monitor	Cloud Operations Suite	AWS is the least expensive amongst [64]
TR 05	AWS CloudWatch	Azure Monitor	Cloud Operations Suite	AWS is the least expensive amongst all[64]

TR	AWS	Azure	GCP	Ranking
TR 07	AWS IAM	Azure Active Directory IAM	GCP IAM	AWS IAM is robust and serverless [62]
TR 13	Network availability will not be provided by single service. This will be achieved by stitching different services and deploying them in different zones.			
TR 16	This will depend on overall architecture selection.			
TR 14	This will depend on overall architecture design.			

3.3 The final selection

Amazon Web Services is the winner over other cloud providers. This is because Amazon provides most of the services required at a low cost. The below table provides the list of services offered by AWS to satisfy selected requirements.

3.3.1 The list of services offered by the winner

1. **Amazon Elastic Compute Cloud** [65]
   Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable computing capacity in the cloud. It provides a reliable, scalable and secure infrastructure on demand. It has a scale capacity within minutes with SLA commitment of 99.99.

2. **AWS Elastic Beanstalk** [66]
   AWS Elastic Beanstalk is an orchestration service to get web applications up and running on AWS. Elastic Beanstalk uses core AWS services such as Amazon EC2, Amazon Elastic Container Service (Amazon ECS), Auto Scaling, and Elastic Load Balancing to support applications that need to scale to serve a large number of users.

3. **Elastic Kubernetes Services** [67]
   Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that can be used to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes.

4. **AWS CodeDeploy** [68]
   CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.

5. **AWS Identity and Access Management** [69]

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. We use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources. By relying on IAM, we can create strong isolation foundations in your system, and reduce the risk of developers unintentionally introducing code that leads to a violation of tenant boundaries.

6. **Amazon CloudWatch** [70]
   Amazon CloudWatch is a monitoring and management service that provides data and actionable insights for AWS, hybrid, and on-premises applications and infrastructure resources. With CloudWatch, you can collect and access your performance and operational data in the form of logs and metrics from a single platform. CloudWatch helps you to monitor your stack (applications, infrastructure, and services) and leverage alarms, logs, and events data to take actions and reduce Mean Time to Resolution (MTTR).

7. **AWS CloudTrail** [71]
   AWS CloudTrail is an AWS service that helps you enable operational and risk auditing, governance, and compliance of your AWS account. Actions taken by a user, role, or an AWS service are recorded as events in CloudTrail. Events include actions taken in the AWS Management Console, AWS Command Line Interface, and AWS SDKs and APIs.

8. **Amazon Relational Database Service** [72]
   Amazon Relational Database Service is a distributed relational database service by Amazon Web Services. It makes it easier to set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

9. **Amazon MemoryDB** [73]
   MemoryDB for Redis is a fully managed, Redis-compatible, in-memory database that delivers ultra-fast performance and Multi-AZ durability for modern applications built using microservices architectures. MemoryDB stores the entire database in memory, enabling low latency and high throughput data access. It is compatible with Redis, a popular open-source data store, enabling you to leverage Redis' flexible and friendly data structures, APIs, and commands.

10. **Amazon Simple Storage Service** [74]
    Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Amazon S3 can be used to store and protect any amount of data for a range of use cases, such

as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.

11. **Amazon DynamoDB** [75]
    Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data

12. **Application Load Balancer** [76]
    A load balancer serves as the single point of contact for clients. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones. This increases the availability of your application.

4 The first design draft

4.1 The basic building blocks of the design

There are different **services** that form basic building blocks of the architecture and each service offers services to satisfy different requirements and all these services together meet most if not all the requirements.

4.1.1 Compute Architecture Selection

Amazon Elastic Kubernetes Service [77]

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that can be used to run Kubernetes on AWS without needing to install, operate, and maintain our own Kubernetes control plane or nodes. We can choose to run our EKS clusters using AWS Fargate, removing the need to provision and manage servers. Managing Amazon EKS is simple due to integrations with AWS Services such as Amazon CloudWatch, Auto Scaling Groups, and AWS Identity and Access Management (IAM).

4.1.2 Database Architecture Selection

Amazon MemoryDB [78]

MemoryDB for Redis is a fully managed, Redis-compatible, in-memory database that delivers ultra-fast performance and Multi-AZ durability for modern applications built using microservices architectures. MemoryDB stores the entire database in-memory, enabling low latency and high throughput data access. It is compatible with Redis, a popular open source data store, enabling you to leverage Redis' flexible and friendly data structures, APIs, and commands.

Amazon DynamoDB [78]

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling. DynamoDB also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data

4.1.3 Network Architecture Selection

Elastic Load Balancers [79]

The AWS ELB is a software-based load balancer which can be set up and configured in front of a collection of AWS Elastic Compute (EC2) instances. The load balancer serves as a single entry point for consumers of the EC2 instances and distributes incoming traffic across all machines available to receive requests.

Amazon Route 53 [79]

Amazon Route 53 is a highly available and scalable cloud DNS web service. It's designed to give developers and businesses an extremely reliable and cost-effective way to route end users to internet applications by translating names, like www.example.com, into numeric IP addresses, like 192.168.2.1, that computers use to connect to each other. Route 53 is fully compliant with IPv6.

4.2 Top-level, informal validation of the design

When a user accesses the application, the request is routed through the layer-4 load balancer. It efficiently manages network traffic, allowing the application to **scale (TR 14)**. We would have load balancers in multiple available zones to make the application **available (TR 13)**.

The request would then be routed to EKS, which would process it. We would host EKS instances in multiple availability zones (AZs) to ensure **high availability (TR 13)** and **elasticity (TR 14)**. Amazon manages the master nodes, ensuring that resource provisioning and application **scaling** are efficient.

For storing votes, we would use the Amazon Redis service. The ability of Redis to handle multiple requests at once boosts voting application **throughput (TR 18)**. In addition, we intend to host Redis in multiple instances to make the system fully **available (TR 18)**. We plan on storing data to DB after aggregation hence **decreasing response time (TR 20)**.

We intend to use Amazon CloudWatch for monitoring and log collection. With this, we would be able to monitor the application's **health using cloud watch plots (TR 13)**

5 The second design

5.1 Use of the Well-Architected framework (Skipped)

5.2 Discussion of pillars

Reliability [80]

The ability of a workload to carry out its intended function accurately and consistently at the anticipated time falls under the reliability pillar. The capacity to use and test the workload over its whole lifecycle is included in this.

A workload's reliability in the cloud is influenced by a number of variables, resilience being the most important. Resilience is the capacity of a workload to recover from disturbances in infrastructure or services, dynamically acquire computing resources to match demand, and handle disruptions like misconfigurations or momentary network problems. Both a commonly used metric to quantitatively quantify resilience and a specific resiliency goal is availability (also known as service availability).

Security [81]

In order to better your security posture, the security pillar explains how to use cloud technologies to protect data, systems, and assets. AWS and the customer both share responsibilities for security and compliance. With AWS running the host operating system, the virtualization layer, and even the physical security of the premises where the service is housed, the operational burden on the user can be lessened thanks to this shared approach.

AWS is in charge of safeguarding the infrastructure that powers all of the services provided by the AWS Cloud. The hardware, software, networking, and facilities that power AWS Cloud services make up this infrastructure.

A customer's accountability will be based on the AWS Cloud services they choose. This specifies how much setup work the customer is required to do as part of their security ob-

ligations.For example, a service such as Amazon Elastic Compute Cloud (Amazon EC2) is categorized as Infrastructure as a Service (IaaS) and, as such, requires the customer to perform all of the necessary security configuration and management tasks. For abstracted services, such as Amazon S3 and Amazon DynamoDB, AWS operates the infrastructure layer, the operating system, and platforms, and customers access the endpoints to store and retrieve data. Customers are responsible for managing their data (including encryption options), classifying their assets, and using IAM tools to apply the appropriate permissions.

Sustainability [82]

Sustainability can be defined as "development that meets the needs of the present without compromising the ability of future generations to meet their own needs." Your business or organization can have negative environmental impacts like direct or indirect carbon emissions, unrecyclable waste, and damage to shared resources like clean water.

When building cloud workloads, the practice of sustainability is understanding the impacts of the services used, quantifying impacts through the entire workload lifecycle, and applying design principles and best practices to reduce these impacts.

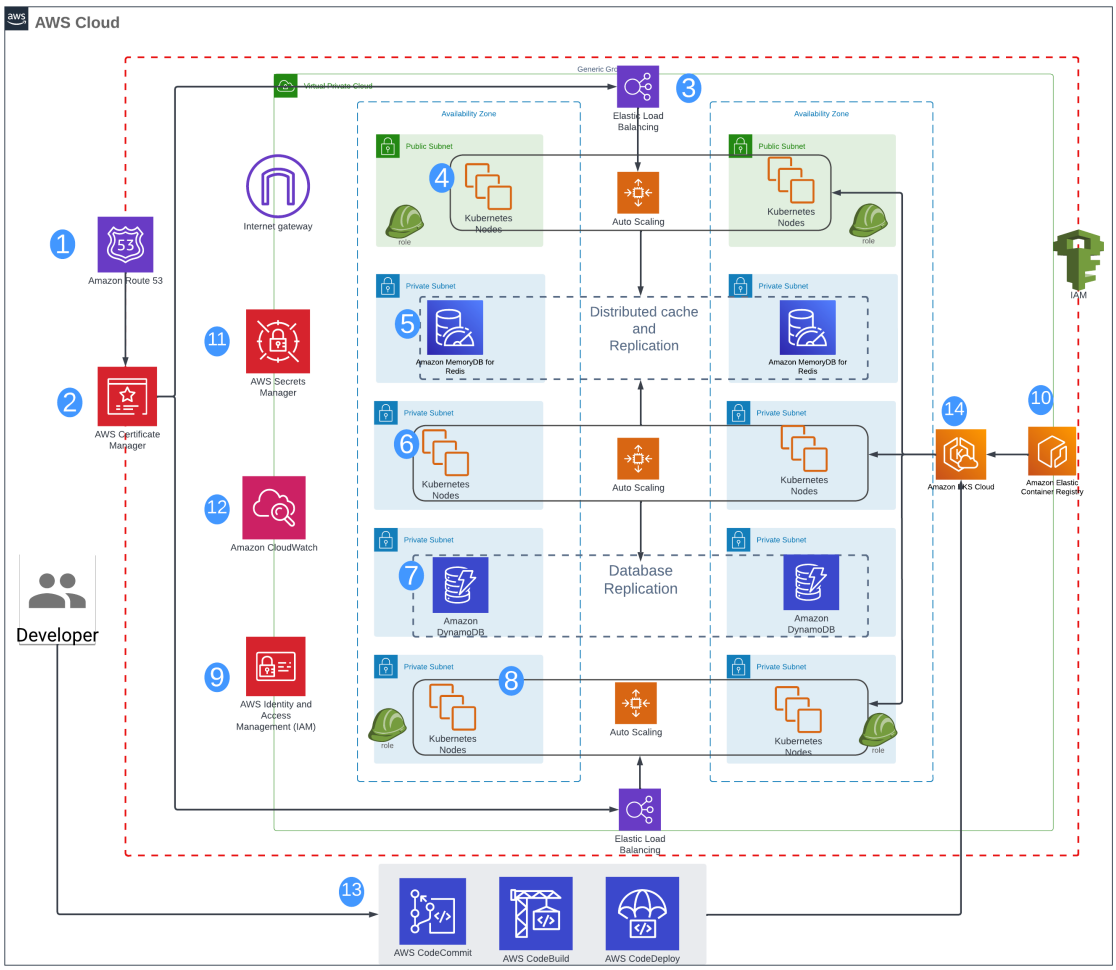
Environmental sustainability is a shared responsibility between customers and AWS.

AWS is responsible for optimizing the sustainability **of** the cloud – delivering efficient, shared infrastructure, water stewardship, and sourcing renewable power. Cloud workloads reduce impact by taking advantage of shared resources, such as networking, power, cooling, and physical facilities. Sustainability in the cloud is a continuous effort focused primarily on energy reduction and efficiency across all components of a workload by achieving the maximum benefit from the resources provisioned and minimizing the total resources required.

Customers are responsible for sustainability **in** the cloud – optimizing workloads and resource utilization and minimizing the total resources required to be deployed for your workloads.

5.3 Use of Cloudformation diagrams

Here is the detailed Cloudformation diagram for Voting app [83]



1. Amazon Route 53 as the DNS service, provides an entry point to the users.
2. Use AWS Certificate Manager to manage your SSL certificates for secure communication with public and private resources.
3. Elastic Load Balancer provides the ability to distribute incoming traffic to compute servers as the target
4. Authenticated traffic from ELB is routed to EKS instances hosting the application logic. These EKS instances are hosted in an auto-scaling group spanning two Availability Zones (AZs) to provide high availability and elasticity, therefore, allowing the application to scale to meet fluctuating demand.
5. Amazon Memory DB collects the new votes. This Memory will be able to take a high concurrent load and decrease the application's latency and enable high throughput data access.
6. A worker consumes votes from Memory DB at it's own pace and store them in a reliable Primary database. These workers are hosted in an auto-scaling group spanning two Availability Zones (AZs) to provide high availability and elasticity,

therefore, allowing the application to scale to meet fluctuating demand.

7. Finally Votes are persisted in the backend Amazon DynamoDB instances. This database layer is configured across multi-AZs, providing high availability and automatic failover.
8. A Front-end web application shows the results of voting in real time quering from the primary databases. These web application serves are hosted in an auto-scaling group spanning two Availability Zones (AZs) to provide high availability and elasticity, therefore, allowing the application to scale to meet fluctuating demand.
9. AWS Identity and Access Management (IAM) Service checks the role for user and grants permission to access the service based on the role defines for the user and action requested by the user.
10. Amazon Elastic Container Registry (ECR) is a fully managed container registry that makes it easy to store, manage, share, and deploy your container images and artifacts anywhere.
11. AWS Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources.
12. Amazon CloudWatch is used for end-to-end platform monitoring through logs collection and application and infrastructure metrics and alerts to support ongoing visibility of the health of the application.
13. Software deployment and associated infrastructure provisioning is automated through infrastructure as code using a combination of Git, Amazon CodeCommit, Amazon CodeBuilt and Amazon CodeDeploy.

5.4 Validation of the design

5.4.1 Comparison between different building blocks

Why Network Load Balancers over Application Load Balancers? [84]

In order to provide high throughput, Incoming traffic needs to be routed between different available application servers. We can either use network or application load balancers for this.

As mentioned above, The AWS NLB is a software-based load balancer which can be set up and configured in front of a collection of AWS Elastic Compute (EC2) instances. This Classic ELB operates at Layer 4. Layer 4 represents the transport layer, and is controlled by the protocol being used to transmit the request. For web applications, this will most commonly be the TCP/IP protocol, although UDP may also be used. A network device,

of which the Classic ELB is an example, reads the protocol and port of the incoming request, and then routes it to one or more backend servers.

The Classic ELB and the ALB share commonalities in function, but the ALB has been specialized to provide users with enhanced capabilities. The ALB operates at Layer 7. Layer 7 represents the application layer, and as such allows for the redirection of traffic based on the content of the request. Whereas a request to a specific URL backed by a Classic ELB would only enable routing to a particular pool of homogeneous servers, the ALB can route based on the content of the URL, and direct to a specific subgroup of backing servers existing in a heterogeneous collection registered with the load balancer.

Since our environment consists of clearly defined services which can each be mapped to a specific address, then the NLB is the logical choice.

Why Amazon Elastic Kubernetes Services over Amazon Elastic Compute Cloud? [85]

EKS is a service that provides and manages a Kubernetes control plane on its own. You have no access to the master nodes on EKS since they're under a special AWS account. To run a Kubernetes workload, EKS establishes the control plane and Kubernetes API in your managed AWS infrastructure and you're good to go. At this point, you can deploy workloads using native K8s tools like kubectl, Kubernetes Dashboard, Helm, and Terraform. This means we don't have to install, operate, and maintain your Kubernetes control plane. With EC2, we have to manually install kubectl, environments etc. EKS comes with inbuilt support for the load balancer of Kubernetes whereas, with EC2, this has to be handled by the hosting personnel. EKS implements VPC by default but with EC2 it is not. EKS gives us fine-grain control about where the containers have to be placed etc along with control over tooling. EC2 is only good if the personnel operating the cluster doesn't have much knowledge about it. From a performance perspective, EKS is much better than hosting Kubernetes on EC2.

Why Amazon CloudWatch over AWS CloudTrail? [86]

CloudWatch is a monitoring service for AWS resources and applications. It helps with Collection and tracking of metrics and monitoring logs of files and setting alarms. CloudTrail is a web service that records API activity in your AWS account. It tells us about who made the request, the service used and the actions performed. CloudWatch helps us in recording the bandwidth, CPU usage, etc whereas the logs by CloudTrail help us in troubleshooting or finding the cause of a security incident, etc.

CloudTrail and CloudWatch can be integrated together to get the most. As we have planned to give importance to monitoring the metrics more than troubleshooting, we are planning to use CloudWatch.

5.4.2 Discussion on how the design satisfies the requirements specified

Amazon Route 53 as the DNS service, provides an entry point to the users. When a user access the application, the DNS resolver resolves the domain name on behalf of the end user and forwards the request to the Route 53. The Route 53 queris the endpoint instance and returns the ip address of the DNS records to the DNS resolver which is sent back to the end user enabling him to access the application smoothly [87]. Route 53 can be setup in minutes and the routing policies can be customized to reduce the **response time (TR 20)**, **improve availability (TR 13)** and **scale automatically (TR 14)**.

Use AWS Certificate Manager to manage your SSL certificates for secure communication with public and private resources. By leveraging this service, the process of obtaining, renewing, and managing the certificates becomes easier. As this is an AWS in-house service there is no additional cost to be paid for using this. Thus **saving money (TR 16)** while **increasing security (TR 09** and **TR 10)**. AWS manages a private certificate authority to ease and faster the certification of internal services. AWS certificate manager establishes secure communication by validating the certificate with the respective Certificate Authority [88].

The request is later forwarded to Layer-4 Load Balancer. This load balancer makes the application **scalable (TR 14)** and **highly available (TR 13)**. The load balancer also monitors the health and performance of the application in real time, and uncovers any bottlenecks while satisfying the SLA [89]. Later, this request is routed to compute server based on the availability and network traffic.

To the EKS instances housing the application logic, load balancer routes authenticated traffic. In order to ensure high **availability (TR 13)** and **elasticity (TR 14)**, these instances are hosted in an auto-scaling group spanning two Availability Zones (AZs). This enables the application to scale to meet fluctuating demand. Elastic Kubernetes Service is managed by Amazon which is easily integrated with in-house AWS services like EC2, VPC, CloudWatch and IAM. The master nodes are managed by Amazon and is efficient in resource provisioning and Kubernetes application scaling while reducing complexity. The Amazon EKS master nodes also manages Amazon EC2 worker instances which are **highly reliable (TR 13)**, scalable, secure compute platforms [90][91].

Amazon Memory DB for Redis serves as in-memory DB which has ultra fast performance. The redis can seamlessly scale from few GB's to TB's to meet the needs of the application. Because of the in-memory db and redis capabilities to handle highly concurrent requests, **the throughput (TR 18)** of the voting application increases. As we are decoupling the voting requests and pushing the data to primary database, **response time of the request decreases (TR 20)** [92]

Votes are persisted in the backend Amazon DynamoDB instances. This database layer is configured across multi-AZs, providing **high availability (TR 13)** and **automatic failover (TR 24)** [93]. Worker nodes aggregates votes as per different criteria and these aggregate results are also stored in Dynamo DB. As Result Declaration application frequently queries the results, we can return the queries with less **Response time (TR 20)**

The AWS **Identity and Access Management (TR 07)** Service verifies the user's role and authorizes access to the service based on the user's defined role and the action they have requested [94].

A fully managed container registry, Amazon Elastic Container Registry (ECR) makes it simple to deploy, share, and manage your container images and artifacts everywhere [95].

You can **safeguard the passwords and API keys (TR 09** and **TR10)** used to access your IT resources, applications, and services with the aid of AWS Secrets Manager [96].

Amazon CloudWatch is used for end-to-end platform monitoring through logs collection, application and infrastructure metrics and alerts to support ongoing visibility of the **health of the application (TR 13)**. CloudWatch analyses the logs and metrics to provide visualized analytics [97].

Amazon CodeCommit, Amazon CodeBuilt, and Amazon CodeDeploy are used in conjunction with infrastructure as code to automate software deployment and the related infrastructure provisioning [98].

The application has mainly 2 tenants. Tenant 1 tries to cast a vote whereas Tenant 2 tries to read the results. The application is developed and deployed such that these two tenants are on different load balancers. Tenant 1 is not affected even if tenant 2 doesn't work and vice versa. To obtain data from each tenant, CloudWatch is used where the traffic of each load balancer is shown separately. So, **TR 26 Tenant Identification and Isolation is achieved**.

5.5 Design principles and best practices used

As a part of architecture, we followed few design principles which suited for this application.

Enable traceability [99]

Monitor,alert,and audit actions and changes to environment in real time. Integrate log and metric collection with systems to automatically investigate and take action.

Scale horizontally to increase aggregate workload availability [100]

Replace one larger source with multiple small resources to reduce the impact of a single failure on the overall workload. Distribute requests across multiple, smaller resources to ensure that they don't share a common point of failure. In this, app instead of using single large application, we divided the application into multiple components, Voting application, Result application and then one more worker and added one in-memory database and primary database.

Go global in minutes [101]

Deploying the workload in multiple AWS Regions around the world allows to provide lower latency and a better experience for your users at minimal cost.

Consider mechanical sympathy [101]

Understand how cloud services are consumed and always use the technology approach that aligns best with the workload goals. For example, we considered data access patterns and since, our data access pattern is always based on single key or aggregate values, we selected Amazon Dynamo DB and since, we need support of concurrent writes with very high throughput, we considered Amazon Memory DB with redis.

5.6 Tradeoffs revisited

5.6.1 TR 18 - Provide High throughput vs TR 20 - Reduced Response time

Before mentioning tradeoffs between Reduce Response time and High Throughput, let us first understand the exact requirements.

Reduce Response time is a "selfish" goal; minimizing response time for individual requests, regardless of the expense to other tasks and requests on the system. Response time is measured as average elapsed seconds.

Whereas High Throughput is a holistic system-wide optimizer goal that is concerned with optimizing your entire workload to maximize total throughput as a whole. High throughput is often associated with parallel requests. Throughput is measured as transactions or requests per second. [102]

The trade-off between throughput and response time becomes evident when you try to balance the ongoing need for high request throughput with an immediate need to perform a large decision-support request like count the number of votes in particular criteria to declare the results for that criteria. The more resources that you apply to the request, the fewer you have available to process other requests, and the larger the impact your query can have on request throughput. Conversely, the fewer resources you allow the request, the longer the request takes. [103]

In our application, we have two main scenarios, Voting scenario and Result declaration scenario. let us analyse each scenario individually.

In Voting scenario, high throughput is expected as multiple people vote at the same time as soon as polling starts. Even though low latency is also expected here, high throughput takes takes top priority. In order to achieve high throughput here we deploy the service in multiple availability zones and elastic load balancer distributes the incoming traffic into different servers therefore allowing thee system to take high concurrent requests. Here we can also meet low latency by decreasing the workload in the scenario, so we decoupled the action of person voting request and sending this vote request to primary database. Instead of the whole action at one place, we decoupled it and the person vote at first goes to Amazon Memory DB redis cache and the request is returned successfully to user does decreasing the workload and thereby decreasing the response time. Then a separate worker pulls these votes from Memory DB and pushes it to Primary Database at it's own pace, thereby achieving both low latency and high throughput at the same time in the Voting Scenario.

Let us consider Result declarations scenario now. In this scenario, the data access pattern is more about read requests. So, inorder to decrease the response time, While worker is pulling the votes and populating the primary database, worker also aggregates the results based on different criteria and populates the aggregate results in database. So, when a user queries for the results, request is served with low response time, as everything is calculated and only the final result is stored in database. As mentioned above high throughput is achieved by maintaining servers in different available zones and routing incoming traffic to different server by elastic load balancer.

In this way, we tried to achieve **high throughput in Voting scenario by trying to reduce response time as much as possible by decoupling workloads** and tried to achieve **reduce response time in Result Declaration scenario by pro-actively doing the aggregates and storing the aggregate results in primary database** and achieving throughput as much as possible by deploying serves in different available zones.

5.6.2 TR 23 - Use Immutable Infrastructure vs TR 24 - Data Backup and Recovery

"Immutable Infrastructure" means, as the name implies, infrastructure that does not change. Once an infrastructure component is provisioned, it is never touched again. If an update, change, or new deployment is required, the existing component is destroyed and replaced by a new one. [104]

Immutability restricts the potential for configuration drift, reducing the IT infrastructure's

vulnerability to attack. Uptime is improved in unexpected events, because instances are redeployed instead of restored from multiple unique configurations and versions.

Immutable infrastructure benefits also include lower IT complexity and failures, improved security and easier troubleshooting than on mutable infrastructure. It eliminates server patching and configuration changes, because each update to the service or application workload initiates a new, tested and up-to-date instance. There is no need to track changes. If the new instance does not meet expectations, it is simple to roll back to the prior known-good instance. Because you're not working with individual components within the environment, there are far fewer chances for unpredictable behaviors or unintended consequences of code changes.[105]

Now the main problem here is What if this application was writing to its local disk and it had data that mattered to the application? Here what we can do is, we can create a new machine, delete that machine, including its data, including its disk. That clearly doesn't work as we lost the data along with the old machine. [106]

To make this effective, what we did is externalize the data. Instead of it being on box with the same application, We used an external database that's shared. So our new machine will write data to the same database that old machine used to write.

As we make this transition, Now, we don't have to worry that we are destroying the data on this box, because We've externalized it. The externalization of data allows the immutable pattern to be applied here. In general, database-like systems tend to be updated much less often than things like our applications, so we gonna use a mutable approach to managing databases because it's so infrequent and we don't have to bother with data migration.

By separating the application to application-tier and data-tier, we maintain immutable infrastructure to only stateless part of the application. When comes to data-tier where we maintain states of different components and different users, we maintain mutable infrastructure.

5.6.3 TR 14 - Implement Scalable and Distributed Architecture vs TR 16 - Choose cost-effective tools and services

The basic aim of scaling a system is to increase its capacity to grow and manage along with increased demand.

Inorder to increase the capacity of the system, we can "scale-up" the system, which is also referred as Vertical Scaling, means adding of more power (CPU, RAM etc.) to your servers. When traffic is low, vertical scaling is a great option, and the simplicity of vertical scaling is its main advantage. Unfortunately, it comes with serious limitations. Vertical scaling has hard limit. It is impossible to add unlimited CPU and memory to single

server. Vertical scaling does not have failover and redundancy. If one server goes down, the website goes down with it completely.

So, we need to come up with new scaling technique which is known as Horizontal scaling also known as "scale-up" mechanism. This allows us to scale by adding more servers into our pool of resources.

In general, when many users access the website simultaneously and it reaches out the web-server's load limit, users generally experience slower response. We can use load balancer to address this problem.

A load balancer evenly distributes incoming traffic among web-server's that are defined in a load-balancer set. **This increases the cost of the system as additional resources like new set of servers and load balancer is required.**

Coming to Data tier, having one database, doesn't support failover and redundancy. Inorder to address this database replication is a common technique. Having a database replication improves performance, reliability and high availability. **By using database redundancy technique cost of the system increases, as we need more resources and a master node to coordinate between different replica nodes.**

Along with database replications, data tier also needs to be scaled with increasing amount of data to be stored. Database can also be scaled using two approaches similar to the above approaches we discussed for our web tier. Vertical and Horizontal Scaling. Vertical scaling of data tier is similar to the vertical scaling we discussed in web-tier, which is achieved by adding more power like CPU, RAM, Disk etc. Horizontal Scaling in data tier is also known as Sharding, is the achieved by adding more serves like we discussed above. [107]

The main difference in Horizontal Scaling in web-tier and data-tier is that in web-tier each server is stateless. It doesn't store any data specific to user, it serves the request irrespective of the state the request is in. But in data-tier, it stores the data as it separates the large database into smaller, more easily managable parts. Data-tier in each shard, store data depending on some property of the data (/user). **As this requires more components and also a master node which controls all the shards, it increases the cost of the system.**

5.7 Discussion of an alternate design (Skipped)

6 Kubernetes experimentation

6.1 Experiment Design

The TRs chosen for Kubernetes Experimentation validation are TR 20 Provide Low Response Time, TR 14 Implement Scalable and Distributed Architecture and TR 15 Implement Container Orchestration, TR 18 - Provide High throughput.

The application is currently deployed into an Amazon Elastic Kubernetes Service. The master nodes for the clusters are managed by the AWS central service. For this implementation, we have chosen to use the Kubernetes auto-scaler. The details of the horizontal scaling algorithm is defined in Kubernetes documentation [108].

The auto-scaler is implemented for all the deployed components. For the demonstration of experimentation, only one component "Voting" is chosen and loaded.

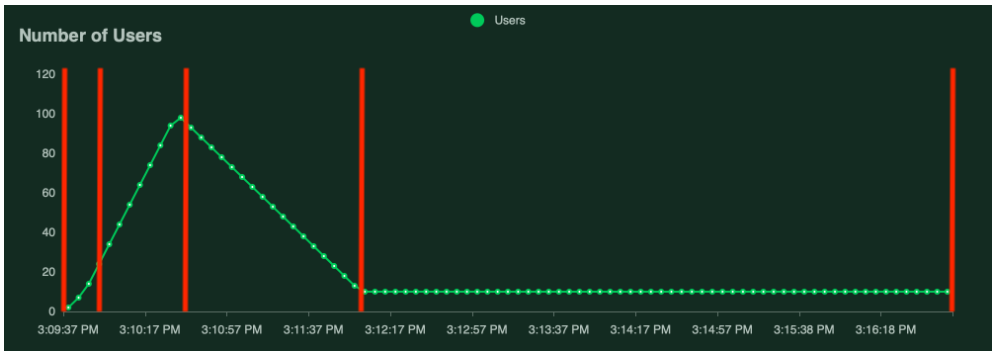
In the experimentation, we will load the application using locust, record and analyze how the auto-scaler works.

6.2 Workload generation with Locust

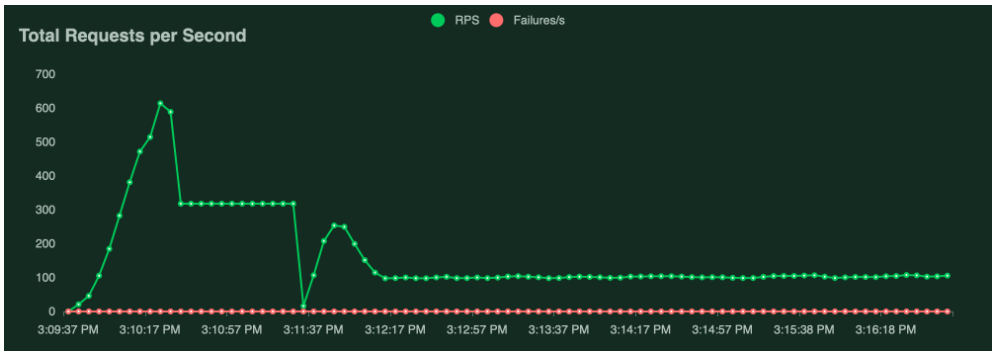
Locust is used to generate load on the hosted service and test it. We have implemented the load in 4 parts.

```
stages = [
    {"duration": 10, "users": 10, "spawn_rate": 1},
    {"duration": 55, "users": 75, "spawn_rate": 2},
    {"duration": 200, "users": 1, "spawn_rate": 1},
    {"duration": 400, "users": 0, "spawn_rate": 1}
]
```

As shown in the figure, in the first part the load is increases the requests very slowly. In the second part, it increases the requests faster than in first stage. In the third stage, the users reduces very slowly. In the fourth part, the users and requests are kept constant and zero.



The first two stages help us in understanding the scale-up functionality and the next two to undertstand the scale-down. The corresponding figure to show number of requests vs time is below.



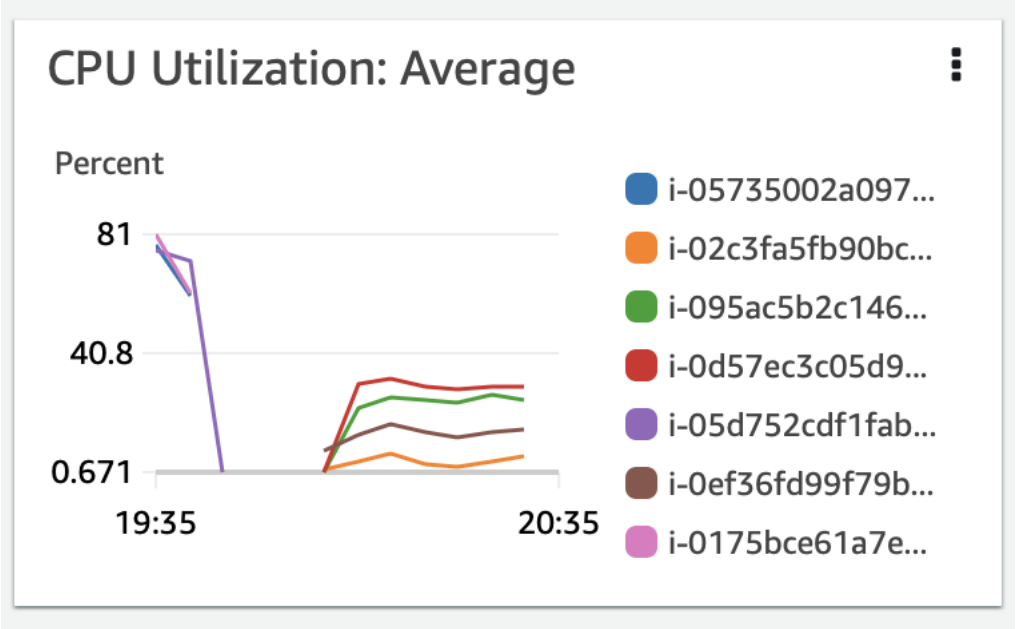
6.3 Analysis of the results

```
(base) ithaaswin@Aswins-Air Course Project % kubectl get hpa --watch
NAME                     REFERENCE                           TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
voting-app-deployment    Deployment/voting-app-deployment    0%/20%     2         30        2          3m4s
voting-app-deployment    Deployment/voting-app-deployment    2%/20%     2         30        2          3m31s     1
voting-app-deployment    Deployment/voting-app-deployment    53%/20%    2         30        2          3m46s
voting-app-deployment    Deployment/voting-app-deployment    99%/20%    2         30        4          4m1s
voting-app-deployment    Deployment/voting-app-deployment    124%/20%   2         30        8          4m16s
voting-app-deployment    Deployment/voting-app-deployment    15%/20%    2         30        13         4m31s
voting-app-deployment    Deployment/voting-app-deployment    0%/20%     2         30        13         4m46s     2
voting-app-deployment    Deployment/voting-app-deployment    5%/20%     2         30        13         5m31s
voting-app-deployment    Deployment/voting-app-deployment    14%/20%    2         30        13         5m46s
voting-app-deployment    Deployment/voting-app-deployment    9%/20%     2         30        13         6m1s
voting-app-deployment    Deployment/voting-app-deployment    6%/20%     2         30        13         6m16s
voting-app-deployment    Deployment/voting-app-deployment    6%/20%     2         30        13         8m32s
voting-app-deployment    Deployment/voting-app-deployment    6%/20%     2         30        13         8m47s
voting-app-deployment    Deployment/voting-app-deployment    6%/20%     2         30        13         9m17s
voting-app-deployment    Deployment/voting-app-deployment    6%/20%     2         30        13         9m32s
voting-app-deployment    Deployment/voting-app-deployment    6%/20%     2         30        10         9m47s
voting-app-deployment    Deployment/voting-app-deployment    7%/20%     2         30        10         10m       3,
voting-app-deployment    Deployment/voting-app-deployment    8%/20%     2         30        10         10m       4
voting-app-deployment    Deployment/voting-app-deployment    7%/20%     2         30        10         10m
voting-app-deployment    Deployment/voting-app-deployment    2%/20%     2         30        6          11m
voting-app-deployment    Deployment/voting-app-deployment    0%/20%     2         30        4          11m
voting-app-deployment    Deployment/voting-app-deployment    0%/20%     2         30        4          15m
voting-app-deployment    Deployment/voting-app-deployment    0%/20%     2         30        2          16m
```

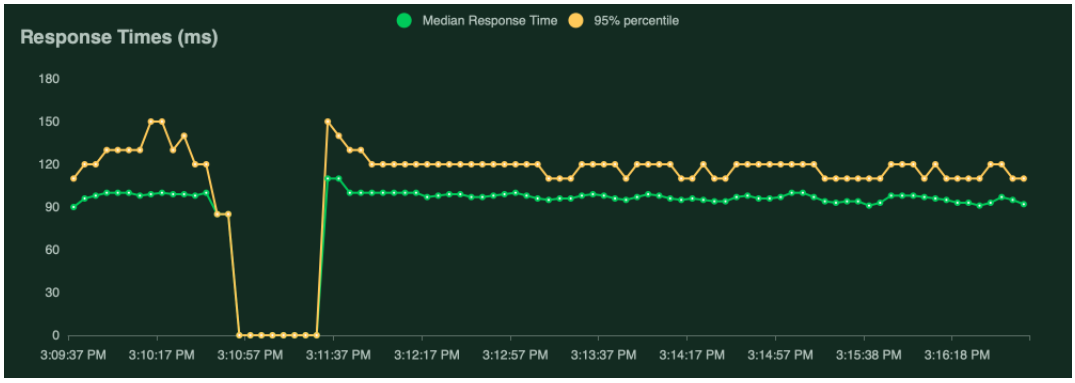
The above picture shows the logs from the autoscaler of the voting-app-deployment. The CPUUtilizationLimit is set to 20%. The minimum number of pods running is 2 and the max limit is 30 pods. Any increase in the utilization more than the set CPU threshold leads to pods scaling up. This can be seen in the first two blocks of the picture. As the load is increased using locust, the cpu utilization increases. Due to this, new pods are spawned and a maximum of 13 pods is reached.

When the number of requests is decreased, the pods are not sccaled down instantly. This is inline with the expected behavior of Kubernetes autoscaler. The autoscaler takes some time to see if the number of requests have truly gone down or if it is just instantaneous and then scales down the pods accordingly. This can be observed in the third block of the above picture. When the number of requests are dropped to almost nill, the number of pods are reduced to the minPods which is 2 in this case.

The CPU Utilization of the worker nodes during this process is as in below picture. The 4 lines are the 4 worker nodes running for the EKS and it can be noted that the cpu utilization of all the worker nodes are mainained to be less than 30%. If there is an increase in the CPU utilization of the worker nodes, EKS is made capable of creating new worker instances.



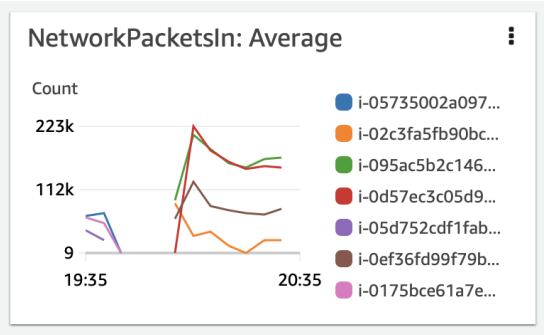
The response time taken to reach the application is defined in the picture below.

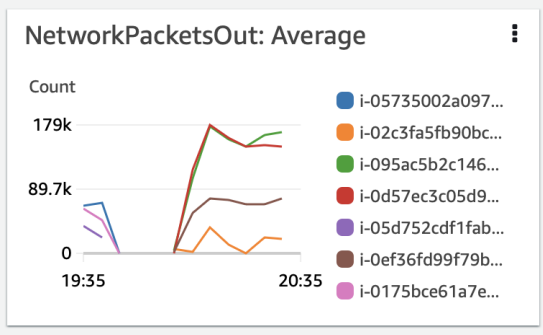


As seen in the picture, there is no drastic difference in the response time of the application. The drop that is seen in between is due to the non availability of data. This is the point where the CPU utilization reaches maxiumum and the new containers are being spawned. During this time, the response time of the application is higher than normal and thus

removed from the plot by locust as outliers. This is expected behavior as new pods take time to spawn and are not instantaneous.

The response time of the application is less than 200ms which is the desired load time as defined in **TR 20**.





The above picture shows details about the packets in and out. The graphs are very similar thus establishing that there are very minimal failures.

7 Ansible playbooks (Skipped)

7.1 Description of management tasks

7.2 Playbook Design

7.3 Experiment runs

8 Demonstration (Skipped)

9 Comparisons (Skipped)

10 Conclusion

10.1 The lessons learned

- We learned about different job roles, the skills required for them, and mainly the difference between them in the field of Cloud Computing.
- We learned about how to gather good architectural requirements (BRs and TRs) and best practices that need to be followed.
- We learned deeply about the 6 pillars of AWS WAF, and why and how it needs to be followed.
- We understood the importance of TradeOffs, and how to compare, evaluate and make decisions between requirements.

10.2 Possible continuation of the project

- For implementation, not all the original TRs were used. The project can be extended during the next phase to implement all of these TRs.
- Our current implementation is done using AWS. This can be made with other famous providers like Azure and a comparison can be made to understand the advantages and disadvantages of this design.
- We have used Amazon Elastic Kubernetes Service which is a PaaS. The same design can be implemented in IaaS and understand the pros and cons of each solution.

11 References

[1] YV and YP, "ECE547/CSC547 textbook and class notes"

[2] "The pillars of the framework" https://docs.aws.amazon.com/pdfs/wellarchitected/latest/framework/wellarchitected-framework.pdf#page=9

[3] OPS05-BP01 Use version control https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=40

[4] OPS05-BP04 Use build and deployment management systems https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=43

[5] OPS06-BP03 Use deployment management systems https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=51

[6] REL13-BP01 Define recovery objectives for downtime and data loss https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=111

[7] REL13-BP02 Use defined recovery strategies to meet the recovery objectives https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=115

[8] REL13-BP03 Test disaster recovery implementation to validate the implementation https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=124

[9] OPS08-BP01 Identify key performance indicators https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=63

[10] OPS08-BP02 Define workload metrics https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=64

[11] REL11-BP06 Send notifications when events impact availability https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=98

[12] REL06-BP03 Send notifications (Real-time processing and alarming) https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=53

[13] SEC01-BP01 Separate workloads using accounts https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=11

[14] COST02-BP03 Implement an account structure https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=27

[15] SEC02-BP01 Use strong sign-in mechanisms https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=8

[16] SEC02-BP03 Store and use secrets securely https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=21

[17] SEC04-BP02 Analyze logs, findings, and metrics centrally https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=36

[18] COST03-BP01 Configure detailed information sources https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=31

[19] SEC05-BP01 Create network layers https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=41

[20] SEC05-BP02 Control traffic at all layers https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=42

[21] SEC06-BP01 Perform vulnerability management https://docs.aws.amazon.com/

pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=46

[22] SEC07-BP04 Define data lifecycle management https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=54

[23] SEC08-BP05 Use mechanisms to keep people away from data https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=59

[24] SEC10-BP01 Identify key personnel and external resources https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=66

[25] SEC10-BP02 Develop incident management plans https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=66

[26] REL01-BP05 Automate quota management https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=17

[27] REL01-BP06 Ensure that a sufficient gap exists between the current quotas and the maximum usage to accommodate failover https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=19

[28] REL02-BP01 Use highly available network connectivity for your workload public endpoints https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=20

[29] Design your workload service architecture https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=29

[30] REL04-BP02 Implement loosely coupled dependencies https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=35

[31] REL03-BP01 Choose how to segment your workload https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=29

[32] OPS04-BP05 Implement transaction traceability https://docs.aws.amazon.com/

pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=38

[33] PERF05-BP04 Leverage load-balancing and encryption offloading https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#page=56

[34] REL10-BP03 Automate recovery for components constrained to a single location https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=85

[35] COST05-BP05 Select components of this workload to optimize cost in line with organization priorities https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=43

[36] COST05-BP04 Select software with cost-effective licensing https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=43

[37] SUS01-BP01 Choose Regions near Amazon renewable energy projects and Regions where the grid has a published carbon intensity that is lower than other locations (or Regions) https://docs.aws.amazon.com/pdfs/wellarchitected/latest/sustainability-pillar/wellarchitected-sustainability-pillar.pdf#page=17

[38] REL10-BP01 Deploy the workload to multiple locations https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=78

[39] REL04-BP02 Implement loosely coupled dependencies https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=35

[40] REL05-BP02 Throttle requests https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=42

[41] REL05-BP03 Control and limit retry calls https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=43

[42] SEC06-BP01 Perform vulnerability management https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=45

[43] REL07-BP01 Use automation when obtaining or scaling resources https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=57

[44] PERF04-BP03 Collect and record database performance metrics https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#page=43

[45] REL02-BP03 Ensure IP subnet allocation accounts for expansion and availability https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=24

[46] PERF05-BP03 Choose appropriately sized dedicated connectivity or VPN for hybrid workloads https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#page=55

[47] REL06-BP02 Define and calculate metrics (Aggregation) https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=51

[48] OPS07-BP03 Use runbooks to perform procedures https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=59

[49] OPS07-BP04 Use playbooks to investigate issues https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=60

[50] OPS05-BP05 Perform patch management https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#page=44

[51] REL08-BP04 Deploy using immutable infrastructure https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=65

[52] REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=69

[53] COST01-BP02 Establish a partnership between finance and technology https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=11

[54] COST01-BP07 Keep up-to-date with new service releases https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=20

[55] COST01-BP08 Create a cost-aware culture https://docs.aws.amazon.com/pdfs/wellarchitected/latest/cost-optimization-pillar/wellarchitected-cost-optimization-pillar.pdf#page=21

[56] "5.5 Tenant collocation" of course Textbook

[57] http://www.dba-oracle.com/t_tuning_high_throughput_vs_fast_response_time.htm

[58] https://www.reblaze.com/blog/devsecops/immutable-infrastructure-and-security/

[59] https://devcontentops.io/post/2021/05/azure-devops-vs-aws-devops-vs-gcp-devops

[60] https://blog.skeddly.com/2020/01/comparing-managed-redis-services-on-aws-azure-and-gcp.html

[61] https://acloudguru.com/blog/engineering/aks-vs-eks-vs-gke-managed-kubernetes-services-compared

[62] https://acloudguru.com/blog/engineering/comparing-aws-azure-and-google-cloud-iam-services

[63] https://acloudguru.com/blog/engineering/storage-showdown-aws-vs-azure-vs-gcp-cloud-comparison

[64] https://medium.com/@richard_64931/monitoring-service-comparison-aws-vs-azure-vs-gcp-part-2-7a9cd52b10f2

[65] https://docs.aws.amazon.com/pdfs/AWSEC2/latest/UserGuide/ec2-ug.pdf

[66] https://aws.amazon.com/documentation-overview/elasticbeanstalk

[67] https://docs.aws.amazon.com/pdfs/eks/latest/userguide/eks-ug.pdf

[68] https://docs.aws.amazon.com/pdfs/codedeploy/latest/userguide/codedeploy-user.pdf

[69] https://docs.aws.amazon.com/pdfs/IAM/latest/UserGuide/iam-ug.pdf

[70] https://aws.amazon.com/documentation-overview/cloudwatch/

[71] https://docs.aws.amazon.com/pdfs/awscloudtrail/latest/userguide/awscloudtrail-ug.pdf

[72] https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide//pdfs/AmazonRDS/latest/UserGuide/rds-ug.pdf

[73] https://docs.aws.amazon.com/pdfs/memorydb/latest/APIReference/memorydb-guide.pdf.pdf

[74] https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf

[75] https://docs.aws.amazon.com/pdfs/amazondynamodb/latest/developerguide/dynamodb-dg.pdf

[76] https://docs.aws.amazon.com/pdfs/elasticloadbalancing/latest/application/elb-ag.pdf

[77] "Compute Architcture Selection" https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#page=15

[78] "Database architecture selection" https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#page=33

[79] "Network architecture selection" https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#page=49

[80] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#page=7

[81] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#page=6

[82] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/sustainability-pillar/wellarchitected-sustainability-pillar.pdf#page=5

[83] https://d1.awsstatic.com/architecture-diagrams/ArchitectureDiagrams/moodle-for-high-availability-on-AWS-ra.pdf

[84] https://www.sumologic.com/blog/aws-elb-alb/

[85] What is EKS? https://cast.ai/blog/aws-eks-vs-ecs-vs-fargate-where-to-manage-your-kubernetes/

[86] https://www.opsramp.com/guides/aws-monitoring-tool/cloudtrail-vs-cloudwatch/

[87] Amazon Route 53 https://aws.amazon.com/route53/

[88] AWS Certificate Manager https://aws.amazon.com/certificate-manager/

[89] Elastic Load Balancing https://aws.amazon.com/elasticloadbalancing/

[90] Amazon EKS https://aws.amazon.com/eks/

[91] Amazon EC2 https://aws.amazon.com/ec2/

[92] Amazon MemoryDB for Redis https://aws.amazon.com/memorydb/

[93] Amazon DynamoDB https://aws.amazon.com/dynamodb/

[94] AWS Identity and Access Management https://aws.amazon.com/iam/

[95] Amazon Elastic Container Registry https://aws.amazon.com/ecr/

[96] AWS Secret Manager https://aws.amazon.com/secrets-manager/

[97] Amazon CloudWatch https://aws.amazon.com/cloudwatch/

[98] Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline https://aws.amazon.com/blogs/devops/complete-ci-cd-with-aws-codecommit-aws-codebuild-aws-codedeploy-and-aws-codepipeline/

[99] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/framework/wellarchitected-framework.pdf#page=17

[100] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/framework/wellarchitected-framework.pdf#page=23

[101] https://docs.aws.amazon.com/pdfs/wellarchitected/latest/framework/wellarchitected-framework.pdf#page=28

[102] http://www.dba-oracle.com/t_tuning_high_throughput_vs_fast_response_time.htm

[103] https://help.hcltechsw.com/onedb/2.0.1/prf/ids_prf_046.html

[104] https://www.reblaze.com/blog/devsecops/immutable-infrastructure-and-security/

[105] https://www.techtarget.com/searchitoperations/definition/immutable-infrastructure

[106] https://www.hashicorp.com/resources/what-is-mutable-vs-immutable-infrastructure

[107] "Chapter 1 Scale from Zero to Millions of Users" from System Design Interview Book by Alex Xu

[108] "Algorithm Details" https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#algorithm-details