Cloud Computing Technology

**CSC/ECE 547 - Fall 2022**

# Intelli-Image

Ashwin Sapre, Chinmaya Srivatsa, Shruti Kohakade
(asapre , csrivat , smkohaka)
North Carolina State University

Plagiarism Declaration

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/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.

# Table of Contents

# 1 Introduction (5%)

## 1.1 Motivation

Our group wanted to pick a topic for the cloud computing project which would allow us to apply the concepts we have learnt throughout the cloud computing class. As far as the application was concerned, we wished to architect for a real-life application, but one which would not take too much time to implement (as the professor has stated multiple times, “this is not a devops class”).

## 1.2 Executive Summary

With the advent of image-sharing apps like Instagram, Pinterest, and to an extent Reddit and 9GAG, there is a high demand for rapid photo editing capabilities before a user uploads an image to the internet. Applications such as Paint are often too basic to achieve most users’ photo editing goals, whereas sophisticated photo-editing usually requires purchase, download and installation of expensive proprietary software such as Photoshop. The performance of such software often suffers on older machines. Similar disadvantages exist on open-source software such as GIMP.

We wanted to create a cloud-based SaaS alternative which could replicate (most of) Photoshop’s image editing features, along with “one click enhance” feature for users who do not necessarily know how photo-editing controls work, but who just want to quickly make their pictures “better”. By making this application cloud-based, we eliminate the requirement of download and installation, remove (or decrease) the cost, and guarantee high performance.

# 2 Problem Description (20%)

## 2.1 The problem

To design a SaaS image manipulation service that allows users to upload an image, perform edits, and securely store the users' data. The service should have minimum latency and provide a desktop-like experience to users.

## 2.2 Business Requirements

- BR 1: Reduce latency while editing images
- BR 2: Be available 24 x 7
- BR 3: Provide high performance
- BR 4: Accommodate time-varying workloads
- BR 5: Reduce costs
- BR 6: Persistently store files uploaded to the service
- BR 7: Securely store user data and files uploaded to the service
- BR 8: Log image manipulation operations and provide the ability to undo/redo changes
- BR 9: Easily integrate with third party applications and software

## 2.3 Technical Requirements

- TR 1: The capacity for CPU, memory, and storage resources should grow or shrink according to the changing demands for our services.
    - New compute instances will be spawned when the average CPU utilization of an existing instance exceeds 75%.
    - Existing compute instances will be terminated when their average CPU utilization goes below 25%.
- TR 2: The availability of the system should be 99%.
- TR 3: Identify a tenant and authorize access before performing any operation.
- TR 4: Obtain performance and cost metrics from monitoring tools.
- TR 5: Ensure a maximum latency of 1 second for all image manipulation operations.
- TR 6: Minimize cost while keeping performance at an acceptable level (as described in TR 5: using elasticity, monitoring and management functions of a cloud environment.
- TR 7: Protect user data from unauthorized access.
- TR 8: Prevent faults from occurring in the system.
    - Provide 99.99% durability to user uploaded images.
    - Recover from failures using logs
- TR 9: Serve multiple tenants at the same time.
- TR 10: Provide the ability to easily maintain and repair the system.
- TR 11: Perform periodic data backups automatically.

- TR 12: Distribute load across multiple servers to increase performance and improve fault tolerance.
- TR 13: Configure the ability to connect to third party applications by scaling microservices that provide application programming interfaces (APIs) for smooth integration.
- TR 14: Store user data, uploaded image data, metadata and edit logs in appropriate storage solutions.
- TR 15: Restrict resource usage to below appropriate limits to prevent cost surges.
- TR 16: Select the most appropriate service offering based on characteristics of data.
- TR 17: Identify patterns of variations in workloads.
- TR 18: Evaluate customer needs.
- TR 19: Evaluate threat landscape.
- TR 20: Obtain workload telemetry.
- TR 21: Identify and prioritize risks using a threat model.
- TR 22: Automate responses to security events.
- TR 23: Log and analyze user actions for better security.
- TR 24: Encrypt sensitive user data and network traffic.
- TR 25: Build services based on functionalities.
- TR 26: Deploy the application to multiple locations.

Revisit justification; revisit all parts of the report in order to make my life easier when grading

| BR(s) | Associated TR(s) | Justification |
|---|---|---|
| BR 1 | TR 1, TR 5, TR 8, TR 12, TR 16 | ● Low latency is our main performance objective, which can only be achieved when the system can accommodate time varying workloads (TR 1, TR 5).<br>● Elasticity, fault tolerance (TR 8) and load balancing (TR 12) would help us achieve that goal.<br>● The selection of appropriate services which are designed to work optimally for a particular task in a cloud environment is important in reducing latency. (TR 16) |
| BR 2 | TR 2, TR 26 | ● We have attached a concrete numerical definition to BR 2.<br>● Providing 99% availability (TR 2) and deploying the application in multiple locations (TR 26) would help us in achieving 24x7 availability |
| BR 3 | TR 1, TR 5, TR 16, TR 17 | ● We have provided a concrete numerical definition of acceptable performance (TR 5).<br>● This performance is linked to elasticity as it ensures that the system adapts to |

| | | |
|---|---|---|
| | | variations in load (TR 1)<br>● These variations must be analyzed to optimize elasticity (TR 17).<br>● To ensure high performance, we must also select the right services from the provider catalog (TR 16) |
| BR 4 | TR 1 | ● To accommodate variations in workload, we need to adjust the amount of resources up or down. This ability of the system is called elasticity. |
| BR 5 | TR 6, TR 15 | ● We aim to minimize costs while keeping performance at an acceptable level (TR 6).<br>● We can prevent cost spikes by implementing resource limits. (TR 15) |
| BR 6 | TR 8 | ● We use the chances of a file being “lost” as a metric to define the persistence of a file. |
| BR 7 | TR 3, TR 7, TR 14, TR 19, TR 21, TR 22, TR 24 | ● We should be able to identify tenants (TR 3) and store each tenant’s files securely and separately in appropriate storage services (TR 14).<br>● To prevent unauthorized access (TR 7) we should identify threats (TR 19, 21), encrypt files and traffic (TR 24), and implement automated measures to security events if they occur (TR 22). |
| BR 8 | TR 3, TR 14 | ● We must identify a tenant (TR 3) and store each tenant’s logs separately and securely in appropriate storage services (TR 14). |
| BR 9 | TR 13 | ● We provide a technical explanation of how third party integration is to be performed. |

### 2.3.1 AWS WAF supply WAF page numbers for each TR

We used the AWS Well-Architected Framework (WAF) [1] as a tool to assist us in the process of creating technical requirements. Here is the list of TRs grouped by pillars of the WAF:

1. Cost optimization: TR 15 (pp 19)
2. Operational excellence: TR 17 (pp 62), 18 (pp 4), 19 (pp 8), 20 (pp 32)
3. Performance efficiency: TR 4 (pp 64), TR 16 (pp 27), TR 12 (pp 49)
4. Reliability: TR 11 (pp 68), TR 25 (pp 26), TR 26 (pp 73)

5. Security: TR 21 (pp 12), TR 22 (pp 33), TR 23 (pp 31), TR 24 (pp 51, 56)

## 2.4 Tradeoffs and Conflicting Requirements

1. [TR 5 conflicts with TR 15]
   The low latency requirement will lead to an increase in the number of server locations where the application is running.
2. [TR 5 conflicts with TR 23]
   Logging all image manipulation operations will increase the storage requirements of the application
3. The elasticity requirement conflicts with cost optimization [TR 1 conflicts with TR 6]. Ideally, we always overprovision resources so that the application never experiences a shortage of compute/storage/network bandwidth. However, doing so would increase costs substantially.
4. [TR 5 conflicts with TR 7]
   Applying security measures degrades the performance of the system.
5. [TR 15 conflicts with TR 5]
   Applying restrictions to resource consumption might lead to reduced performance.
6. [TR 12 conflicts with TR 8]
   Running the application across different regions increases the chances of any one service being affected by server outages.
7. [TR 5 conflicts with TR 24]
   Encryption of sensitive user data and network traffic might increase the time of network transmission which conflicts with the requirement that a maximum latency of 1 second for all image manipulation operations.
8. [TR 6 conflicts with TR 24]
   The cost of encryption of sensitive user data and network traffic is high which conflicts with the requirement that cost of operations should be low while keeping performance at an acceptable level
9. [TR 23 conflicts with TR 6]
   Logging and analysis of user actions will need extra storage and compute resources, which conflicts with the need to minimize the cost of the cloud environment.
10. [TR 13 conflicts with TR 15]
    Performing periodic data backups would increase storage requirements.
11. [TR 13 conflicts with TR 6]
    Performing periodic data backups would also increase cost requirements.

# 3 Provider Selection (20%)

## 3.1 Criteria for choosing a provider

We define a few criteria to help with the selection of the cloud provider:

1. Service catalog: the provider should have all the services that we need to achieve our TRs.
2. Reputation: the provider should have a good reputation in the IT world. It should not be infamous for outages, data leaks, etc.
3. Familiarity: The group members should all be comfortable working with the cloud provider.
4. Pricing: the provider should offer the most competitive pricing for the services we want.
5. Locations: the cloud provider should have a worldwide presence.

## 3.2 Provider Comparison

The following table provides a comparison [2] of the "big 3" cloud providers with respect to the above criteria. We give each provider a score from 0 (worst) to 2 (best).

| ***Provider*** | ***AWS*** | | ***GCP*** | | ***Azure*** | |
|---|---|---|---|---|---|---|
| ***Parameter*** | **Justification** | **Score** | **Justification** | **Score** | **Justification** | **Score** |
| **Service catalog** | Has the largest service catalog (200+) | 2 | Has the smallest service catalog out of the big 3 | 0 | Has the second biggest catalog | 1 |
| **Reputation** | Has the highest market share among the big 3 providers, with clients like Netflix, Expedia, SAP etc. | 2 | Smallest of the big 3, but has clients like Twitter, Spotify | 2 | Second only to AWS in terms of market share; preferred by companies using Microsoft tools | 2 |
| **Familiarity** | Highest familiarity and experience | 2 | Basic experience and knowledge of services | 1 | No familiarity or experience | 0 |
| **Pricing** | AWS offers a 41% discount for a 1 year commitment. AWS still works out to be cheaper than the other two. | 2 | GCP offers a 63% discount for a 1 year commitment. | 1 | Azure offers a 41% discount for a 1 year commitment. | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Locations** | 30 regions and 96 availability zones across all continents | 1 | 35 regions and 106 zones across all continents except Africa | 0 | 60+ regions with multiple availability zones per region across all continents | 2 |

## 3.3 The final selection

The services offered by the big 3 are generally similar. However, since AWS has attractive pricing models, a large host of services in their catalog and especially since we are more familiar with AWS compared to the other cloud providers, we will go with AWS. Hence, it also accounts for the highest cumulative score.

## 3.4 The list of services offered by the winner

a. Amazon S3 - https://aws.amazon.com/s3/

Amazon's Simple Storage Service or S3 is a block storage service. It is capable of storing "objects" of sizes up to 5TB, which can be images, videos, text files, etc. These objects are stored in buckets. AWS promises 11 9s of durability for S3 along with a host of other features, such as:

- Security: Access to buckets and objects inside buckets can be controlled using access control lists and bucket policies.
- Replication: Using S3 replication, S3 objects can be replicated across AWS regions for greater redundancy.
- Storage classes: Based on frequency of access, AWS can automatically segregate your data into tiers (frequent, infrequent, archive, deep-archive) for a fixed monthly monitoring cost.
- Batch operations: AWS allows you to perform operations on billions of objects efficiently with S3 Batch Operations.

b. DynamoDB - https://aws.amazon.com/dynamodb/

It is a managed key-value database service offering millisecond to microsecond latency. DynamoDB has features like global replication, auto-scaling, encryption at rest, and integration with other AWS services.

c. Elastic Compute Cloud (EC2) - https://aws.amazon.com/ec2/

EC2 offers compute instance types for a variety of applications. It supports the latest Intel, AMD and Arm processors, allows cost savings through per-second billing and CPU optimization, and makes your application elastic via its autoscaling features.

d. Elastic Kubernetes Service - https://aws.amazon.com/eks/

It is a managed Kubernetes service. Containers can be run in AWS EC2 or Fargate to make the application highly scalable.

e. Elastic Load Balancing - https://aws.amazon.com/elasticloadbalancing/

It distributes traffic across a set of target IP addresses. ELB offers 3 variants; application load balancing, network load balancing and gateway load balancing.

f. Virtual Private Cloud - https://aws.amazon.com/vpc/

   It is a service that lets you create a logically isolated virtual environment. Using VPCs, you can control inbound and outbound traffic, separate different components of your application, enhance security via security groups, manage internal IP addressing, and set up load balancing.

g. CloudWatch - https://aws.amazon.com/cloudwatch/

   CloudWatch is a logging and monitoring service. It allows users to store, access and analyze logs from several different services in the AWS catalog.

h. Identity and Access Management (IAM) - https://aws.amazon.com/iam/

   With IAM an organization can define who or what can access which services in a centralized, efficient manner.

i. Amazon Relational Database Service - https://aws.amazon.com/rds/

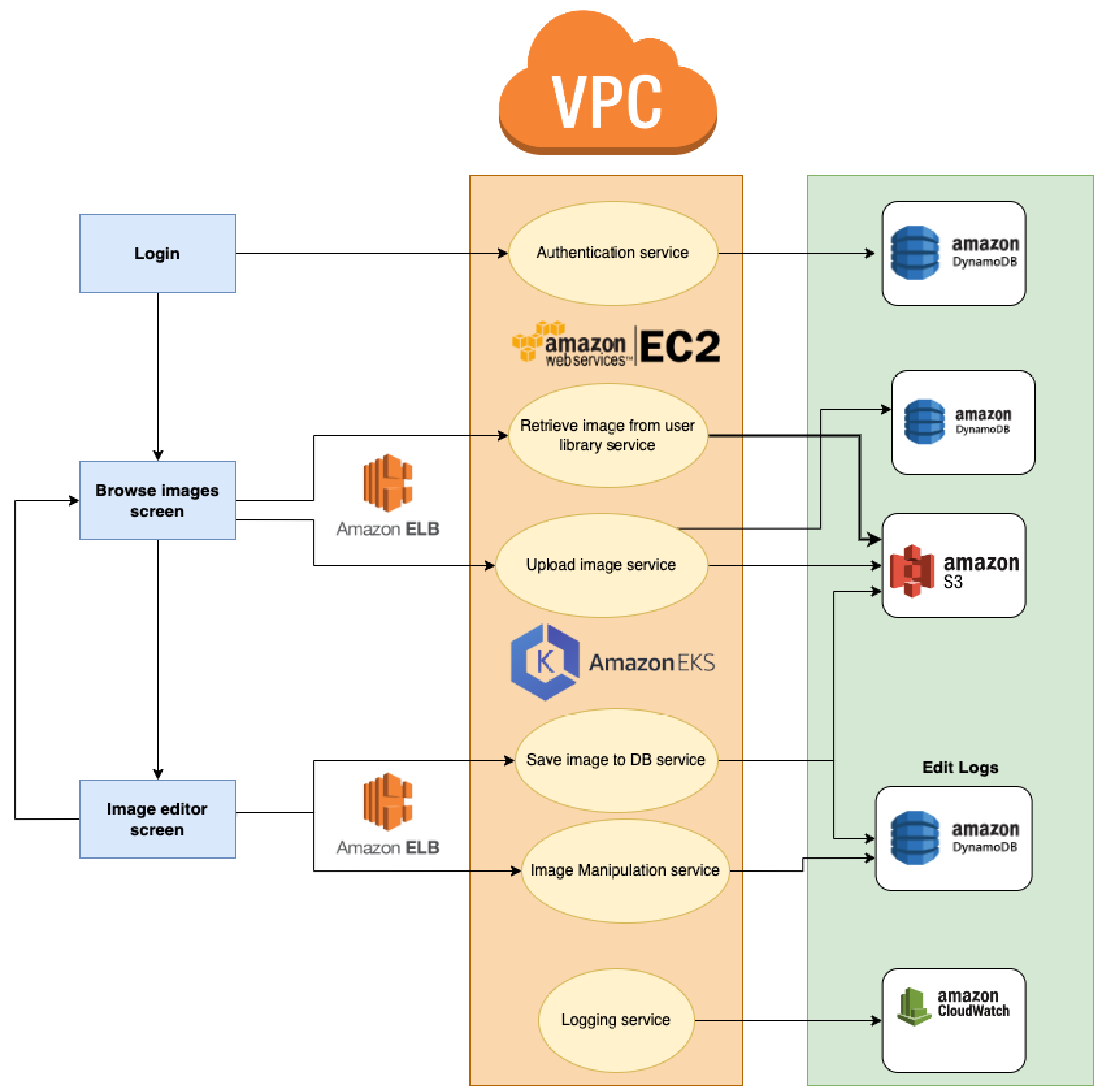
   Amazon Relational Database Service makes it easier to set up, operate,and scale a relational database in the cloud.

# 4 The first design draft (20%)

We select the following TRs from the list supplied in section 2.4.

1. The capacity for CPU, memory, and storage resources should grow or shrink according to the changing demands for our services.
    a. New compute instances will be spawned when the average CPU utilization of an existing instance exceeds 75%.
    b. Existing compute instances will be terminated when their average CPU utilization goes below 25%.
2. The availability of the system should be 99%.
3. Identify a tenant and authorize access before performing any operation. (Tenant identification TR)
4. Obtain performance and cost metrics from monitoring tools (Monitoring TR)
5. Ensure a maximum latency of 1 second for all image manipulation operations. (conflicts with TR 6)
6. Minimize cost while keeping performance at an acceptable level using elasticity, monitoring and management functions of a cloud environment.
7. Restrict resource usage to below appropriate limits to prevent cost surges. (TR from WAF: Performance Efficiency)
8. Prevent faults from occurring in the system (TR from WAF: Reliability)
    a. Provide 99.99% durability to user uploaded images.
    b. Recover from failures using logs
9. Deploy the application to multiple locations. (TR from WAF: Reliability)
10. Distribute load across multiple servers to increase performance and improve fault tolerance. (TR from WAF: Performance Efficiency)
11. Store user data, uploaded image data, metadata and edit logs in appropriate storage solutions. (TR from WAF: Performance Efficiency)
12. Protect user data from unauthorized access. (TR from WAF: Security)
13. Encrypt sensitive user data and network traffic (TR from WAF: Security)

We present a preliminary version of the architecture below:



## 4.1 Basic building blocks of the design

1. Storage
    a. AWS S3: object storage for images.
    b. AWS DynamoDB: for storing user details and image manipulation logs.
2. Compute
    a. Amazon Elastic Kubernetes Service (EKS): EKS handles the Kubernetes Control Plane in an AWS VPC (which means, the client does not have to worry about scaling, backups, patching etc)

    b. Amazon EC2: used together with EKS to run Kubernetes worker nodes in the data plane.
3. Networking
    a. Amazon Elastic Load Balancer: to distribute load across multiple targets. The Application Load Balancer variant uses layer 7 load balancing.
    b. Amazon VPC: to set up public and private subnets, perform ingress routing and internal IP addressing. An ELB can run together with a VPC to distribute traffic across subnets in multiple availability zones. [3]
4. Others
    a. AWS CloudWatch: for storing, managing and analyzing logs from AWS services.
    b. AWS IAM: for identity and access management

## 4.2 Top-level informal validation

1. Storage:
    a. Storing images: Amazon S3 is the optimal choice for this purpose as it is of an “object storage” type. Images uploaded by the user are the most important type of data for the system, since it will need to be accessed the most frequently, replicated to multiple regions, backed up, etc.
    b. Storing metadata: Amazon DynamoDB is the optimal choice as image parameters are best stored as key-value pairs rather than in a relational database.
    c. Storing edit logs: We will store these in DynamoDB as it offers fast retrieval times. This is essential because the undo/redo functionality would need to be as fast as possible. Also, the structure of these logs would be more suited to a key-value store rather than a relational database.
2. Performance:
    a. Latency: Performing undo or redo operations within 1 second is possible since AWS DynamoDB guarantees millisecond times [4] for retrieval.
    b. Elasticity: EKS connects to a data plane, consisting of managed EC2 node groups, accessible by simple API calls. These groups have auto-scaling features which provide elasticity.
3. Networking:
    a. Load balancing: Amazon Elastic Load Balancer provides Application layer load balancing (layer 7) [5].
4. Security:
    a. Encryption: Both S3 and DynamoDB provide encryption at rest [6, 7]. Network traffic inside AWS systems is also encrypted [8].

## 4.3 Action items and rough timeline

Skipped

# 5 Second design draft (20%)

## 5.1 Use of the Well-Architected Framework

Skipped

## 5.2 Discussion of pillars

The AWS Well Architected Framework (WAF) [1] is a guide designed to help people build efficient, reliable, secure, cost-effective and sustainable cloud infrastructure on AWS. It has been created by AWS Solution Architects with several years of experience. The WAF provides best practices, questions that a business should think about, and methodologies that it should follow to extract the full potential out of a cloud architecture. It consists of 6 pillars:

1. Operational Excellence
2. Performance Efficiency
3. Cost Optimization
4. Reliability
5. Security
6. Sustainability

We now describe 3 pillars in detail:

a. ***Performance efficiency* [9]**:

   The performance efficiency pillar describes how a company should use cloud computing resources to consistently and efficiently meet requirements. AWS says there are 4 components to this process:

   1. Selection
   2. Review
   3. Monitoring
   4. Tradeoffs

   **Selection**: All major cloud providers generally have a long list of service offerings. It is critical that clients of these providers have complete knowledge of the catalog, so that they can find a service that most closely matches their needs in an economical manner. Broadly, this selection process is applied to architectural decisions, compute services, storage services, database services and networking services. A few general steps to follow while making service selections are:

   - Understand requirements
   - Evaluate all relevant services
   - Collect usage metrics

**Review:** New services and innovations are released constantly by cloud providers. It is important to stay on top of these new developments and incorporate them into the existing architecture, if an improvement can be made. At the same time, a well-defined process must be followed before deciding to use a new service. A change that is not well-thought out might prove detrimental both from a cost and performance perspective.

**Monitoring**: Monitoring is essential to ensure performance levels stay at expected levels. There are 4 components to the monitoring process:

- Establish key metrics or KPIs
- Record metrics
- Analyze metrics over a period of time
- Set up automated triggers and alarms

**Tradeoffs**: In an ideal world, everything about a given cloud architecture would be perfect; unlimited compute, storage, bandwidth, budgets etc. However, in reality, a company has to choose what features of their architecture are more essential than others, and accordingly prioritize those components. The following strategies may be used while evaluating tradeoffs:

- Researching common design patterns
- Identifying how tradeoffs impact customers and efficiency
- Measuring the impact of architectural changes

### b. *Operational Excellence* [10]

The Operational Excellence pillar helps to apply best practices in the design, delivery and maintenance of AWS workloads. This pillar includes how your organization helps your business objectives, your ability to run workloads effectively, gain insight of the operations, and to continuously try to improve supporting processes and procedures to accomplish business value.
There are five design principles for operational excellence in the cloud:

1. Perform operations as code
2. Make frequent, small, reversible changes
3. Refine operations procedures frequently
4. Anticipate failure
5. Learn from all operational failures

**Perform operations as code:** In the cloud, you can apply the same engineering discipline to your whole environment as you use it for application code. You can define your entire workload (infrastructure, applications, etc) as code. With operations as code you can script the operations and automate their execution by triggering events. By using performing operations as code, you can minimize human error and allow consistency in the operations.

**Make frequent, small, reversible changes:** Design your workloads in such a way that allows the components to improve regularly with useful changes to the workload. Changes should be made in small increments so that they can be reversed in case of failure with zero downtime of your system.

**Refine operations procedures frequently:** As you use operations procedures, you should try to improve as you improve your workload. There should be provision to review and validate the effectiveness of the procedures. The teams should be familiar with this.

**Anticipate failure:** There should be provision to identify potential sources of failure so that they can be removed before any system disaster. Testing of failure scenarios and validation of your understanding of their impact is important. Ensure that your response procedures for these failures are effective and that teams are familiar with their execution. Test these simulated failure events to test workload and see team responses.

**Learn from all operational failures:** Make improvements in the applications and infrastructure through the lessons learned from operational events and failures. This should be shared with the entire organization.

AWS says that there are four ways to achieve operational excellence in the cloud:

- Organization
- Prepare
- Operate
- Evolve

**Organization:**
In order to enable operational excellence, understanding of your organization's aims, your organizational structure, and how your organization backs your team members, so that they can support your business objectives is important.

**Prepare:**
To prepare for operational excellence, the understanding of workloads and their expected behaviors are important. In order to prepare for operational excellence, following needs to be executed:

- Design Telemetry
- Design for operations
- Lower deployment risks
- Operational readiness and change management

**Operate:**
In order to achieve success, understanding of the health of your workload and operations must be there. This can be done by having understanding of workload, operational health, and response to planned and unplanned events.

**Evolve:**
In order to evolve your system i.e. to improve the system, learning from the operations activities, sharing them across the teams and making frequent small incremental changes are necessary.

c. ***Reliability* [11]**

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This includes the ability to operate and test the workload through its total lifecycle. AWS says that these are the best practice guidance for implementing reliable workloads on AWS:

1. Automatically recover from failure
2. Test recovery procedures
3. Scale horizontally to increase aggregate workload availability
4. Stop guessing capacity
5. Manage change in automation

**Automatically recover from failure**: We make use of key performance indicators (KPIs) to monitor workloads and can trigger automation when a threshold is breached. We need to use KPIs that measure business value and not the technical aspects of the operation of the service. This allows for automatic notification and failure tracking, and for automated recovery processes that work around or repair the failure. It is also possible to anticipate and remediate failures before they occur using sophisticated automation.

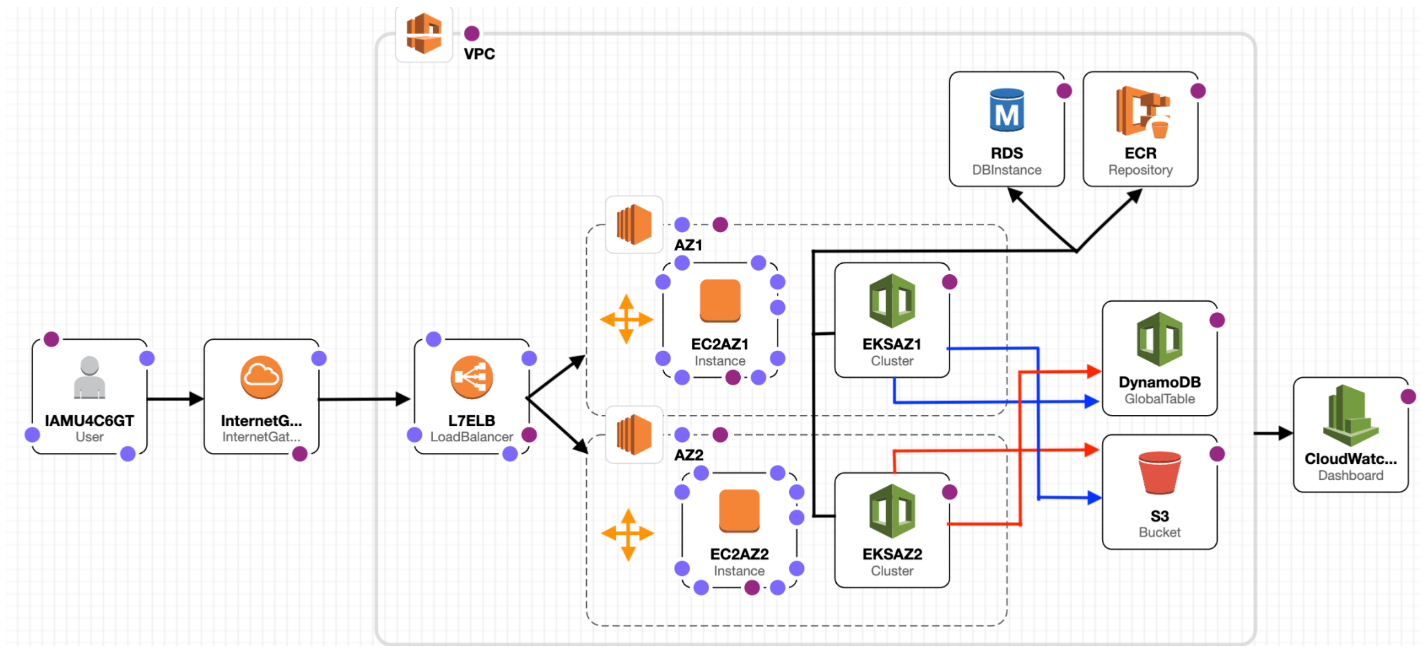
**Test recovery procedures**: Typically testing is often conducted to prove that the workload works in a particular scenario in an on-premise environment. Testing is not typically used to validate recovery strategies. In the cloud however, we can test how our workload fails, and we can validate our recovery procedures. We can simulate different failures and recreate scenarios that led to failures before using automation. This approach exposes failure pathways that we can test and fix before a real failure scenario occurs, thus reducing risk.

**Scale horizontally to increase aggregate workload availability**: We should replace one large resource with multiple small resources to reduce the impact of a single failure on the overall workload. Distributing requests across multiple, smaller resources would ensure that they don't share a common point of failure.

**Stop guessing capacity**: A common cause of failure in on-premises workloads is resource saturation, when the demands placed on a workload exceed the capacity of that workload (e.g. DDOS attacks). In the cloud, we can monitor the demand and workload utilization. We could also automate the addition or removal of resources to maintain the optimal level to satisfy demand without under-provisioning or having excess provisions to the point of redundancy. AWS has still has some limits, but some quotas can be controlled and others can be managed

**Manage change in automation**: Changes to our infrastructure should be made using automation. The changes that need to be managed include changes to the automation, which then can be tracked and reviewed.

## 5.3 Use of CloudFormation diagrams



## 5.4 Validation of the design

*More detailed arguments for why the services we have chosen achieve our TRs? Maybe the metrics they offer? Some features? Guaranteed x% availability/durability etc?*

We now justify why our design allows us to meet the selected TRs in section 4.

1. We collect CPU utilization metrics from EC2 instances where EKS worker nodes are running. In the YAML file, we define criteria to create nodes in our deployment. Then we deploy the Horizontal Pod Autoscaler which creates new pods in the nodes when the mean CPU utilization goes above 60%, and terminates pods using this formula - **desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]**

2. The following strategies [11, pp 118] help us with our goal of 99% availability:

a. Monitoring the availability of the website and creating an alert if something other than an HTTP 200 is returned.
b. Monitoring images stored in S3 for availability.
c. Deploying our application to multiple regions and availability zones for greater resiliency.
d. Autoscaling EC2 instances as described in point 1 to adapt to time-varying workloads.
e. Frequent data-backups for recovery in case of a failure. Recovery procedures use backups from the same region.
f. AWS ELB directs the traffic to kubernetes services. Health checks/probes are run in order to determine when to restart the containers and are used by kubernetes services and deployments to determine if a pod should receive traffic or not.

   99% availability affords us 3.65 days of downtime in a given year. Assuming that we have 5 failures in a year which require manual intervention, and each such failure takes 1hr to recover from. We also assume that 20 failures happen each year from which the system recovers automatically, taking 20 mins per failure.

   Therefore the total downtime is 5 hrs + 400 mins = 11.6667 hrs. In this way we achieve our goal of 99% availability.

3. AWS allows for tenant identification and isolation (Identity Management, User Authentication, and Authorization) along with ability to have some central policies and hence, we can optimize for tenant needs [12][13]
4. AWS Cloudwatch will help us achieve this requirement. Cloudwatch has a number of metrics such as CPU utilization, number of input/output packets, status checks, etc. The organization can configure alarms or set up a mechanism to add/remove resources if the values of these metrics go above or below a certain threshold.
5. We ensure a maximum latency of 1 second for all image manipulation operations by using multithreaded APIs. Amazon EC2 service supports multithreading, which enables multiple threads to run concurrently on a CPU core. By using multithreading each API call and processing will be fast.
6. In order to minimize cost, AWS cloudwatch service provides monitoring and management function that provides data and actionable insights for hybrid, AWS and on-premises applications and infrastructure resources. By using AWS cloudwatch, we can get the statistics based on the metric data point provided by other AWS services and our custom data and this will help in reducing the cost of operations and resources.
7. In order to restrict resource usage to limit the cost, AWS Elastic Kubernetes Service(EKS) can be used. This EKS provides the functionality of kubernetes horizontal pod autoscaler(HPA) which increases the resources when there is high demand for the service and decreases the resources when there is less demand for our services. This is customizable so we can change the configuration according to our needs. This way, by using the autoscaler feature, we can prevent the cost surges.
8.

    a. AWS S3's SLA guarantees 11 9s of durability over a given year.
    b. We take frequent backups of the system, and use them to recover from failures.
9.
    a. S3, DynamoDB are deployed across availability zones [11, pp 75).
    b. EC2 instances in an auto-scaling group that spans multiple availability zones provides greater availability and reliability. When an auto-scaling group is created, we must choose the VPC and subnets it covers. Therefore, every instance belongs to a unique AZ. EC2 automatically balances the number of instances per AZ [14]
10. We use AWS Elastic Load Balancing to achieve this goal. Once the request arrives at a datacenter, we use HTTP endpoints as the basis of load balancing. Through this method we can scale individual microservices separately.
11. We use a variety of storage solutions:
    a. S3 (object storage) to securely store images.
    b. DynamoDB, to store edit logs, metadata and user details.
    c. CloudWatch, to store, organize and analyze logs.
12. We use Cloud IAM to enforce access control in our organization. Attribute-based access control provides fine-grained permissions based on user attributes, such as team, department, role etc [15]. The level of access can also be configured. This prevents unauthorized people from within the organization accessing user data.
13. Several AWS services provide us the ability to encrypt data:
    a. S3 and DynamoDB have built-in encryption at rest [6, 7]
    b. All network traffic within VPCs is encrypted [8].
    c. AWS automatically encrypts data flowing between EC2 instances. [8]

## 5.5 Design principles and best practices used

We use the following design principles taken from the AWS WAF:

1. Go global in minutes ([10], page 2(6)): we use the worldwide presence of AWS to deliver low-latency applications to customers in a cost-effective manner.
2. Scale horizontally to increase aggregate workload availability ([11], page 2(7)): we use several smaller resources and distribute load rather than using one high-capacity resource.
3. Stop guessing capacity ([11], page 2(7)): we use metrics to accurately judge the amount of resources needed to satisfy demand and deliver high quality customer experiences.
4. Protect data in transit and at rest ([16], page 2(6)): we use built-in encryption of data and network traffic provided by AWS services.
5. Perform operations as code ([17], page 2(6)): we automate operations such as load balancing, auto-scaling and responses to security events for high consistency, faster reaction times and low error rates.

## 5.6 Tradeoffs revisited

IMPORTANT - arguments for choosing what tradeoffs you made for conflicting TRs - google tradeoffs decision.
Pick one set of conflicting TRs - explain which TR you chose to optimize. Why and how.

1. [Logging requirement conflicts with increased storage costs]
   In any image editor, undo/redo are fundamental features. Especially when users are new to the application, they will spend some time trying out new filters, presets and other image manipulation operations. This will inevitably lead to mistakes. Therefore the ability to undo/redo are non-negotiable requirements, which is why we choose to tradeoff a better user experience with increased storage requirements.
2. [Data backups will increase storage requirements]
   Our goal of 99% availability is dependent on recovering from failures as quickly as possible. High availability will lead to greater customer satisfaction, and the increased storage requirements is a cost worth paying, because downtimes might force our users to switch to another application.
3. [Setting resource limits might affect performance]
   On one hand, we should aim to deliver the best possible experience to our customers (as stated in the above tradeoffs), even at the cost of higher resource bills. However, we cannot let the bill increase indefinitely. It is necessary to set an upper limit on the amount of resources being consumed and generate alerts when this limit is reached. This process offers another benefit: the organization can identify what resources are being over-consumed, and at what times. Accordingly, it can either revise their limits or search for faults in their architecture which might be causing this overconsumption. Due to these reasons, we sacrifice the ability to accommodate extreme workloads in favor of a resource-limiting mechanism.
4. [Workload telemetry requires extra storage which conflicts with the restricting storage usage]
   In order to make improvements in our application and to improve user experience we need to identify the failures in our system and to mitigate them the workload telemetry data is necessary. This workload telemetry helps in improving the quality of our product and hence in turn attract more customers. So we choose workload telemetry over restricting storage usage.
5. [Cost of encryption of userdata and network traffic conflicts with the requirement of keeping the cost of operations low]
   Encryption of all userdata and network traffic, essentially encryption of every operation is necessary in order to prevent data theft and violation of privacy. Image data of a user is very sensitive and our application must ensure that the user data is protected against every security attack. In order to achieve this, encryption of userdata and network traffic is necessary. Hence we choose to tradeoff cost of operations with encryption of user data and network traffic.
6. [Encryption of userdata and network traffic increases the latency which conflicts with the requirement of maximum latency of 1 second for operations]

The encryption of user data and network traffic is non negotiable to tradeoff for low latency. This is because the major concern in today's world is for data protection and privacy. There are other ways like using powerful compute engines, multithreading to decrease the latency in our operations. So we choose encryption of user data and network traffic over decreased latency because latency can be decreased by other solutions.

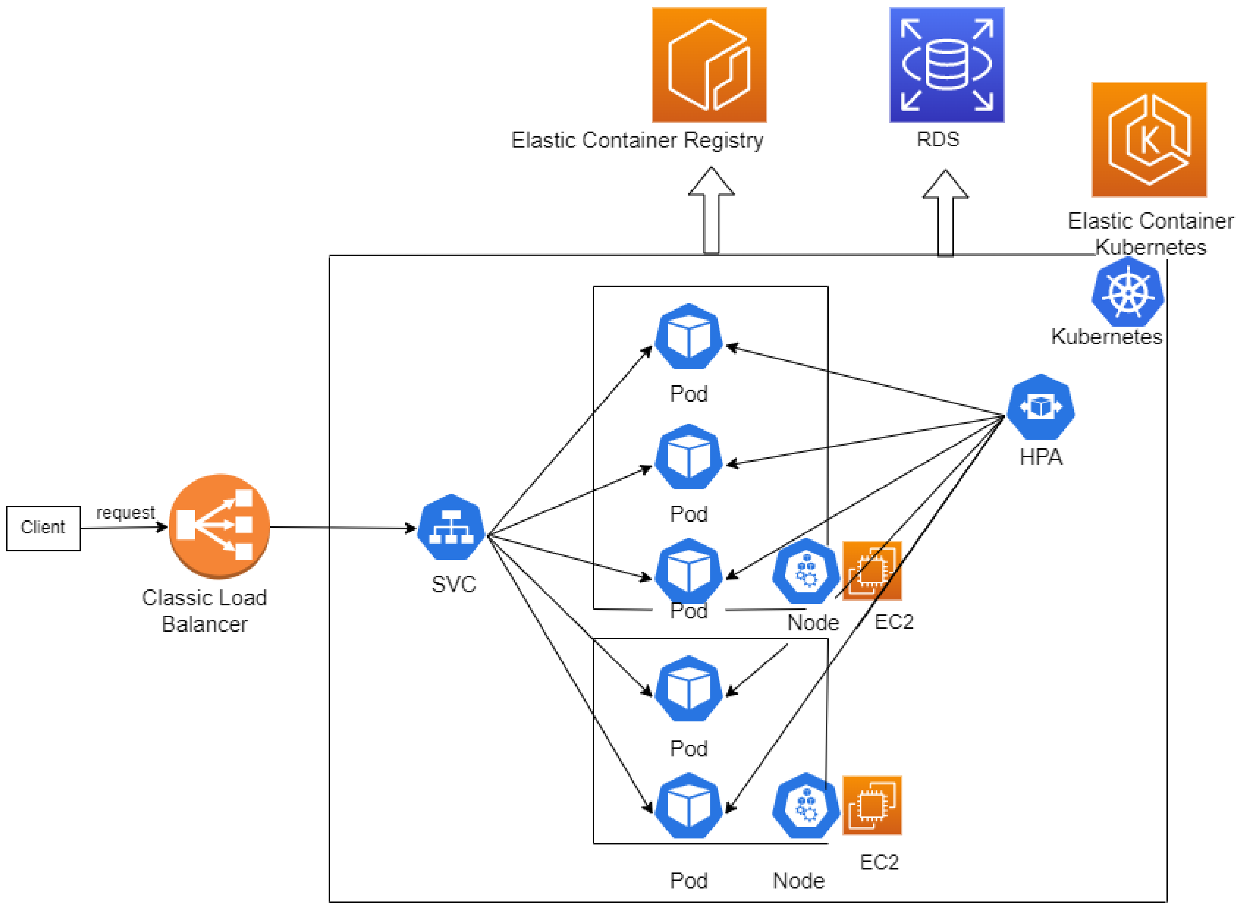
7. [Applying security measures degrades the performance which conflicts with the requirement of high performance of our system]
   In today's world, security is the primary concern for all the users. Hence, authentication and authorization must be done before any operation in our system. Use of security softwares might degrade the performance but they are essential to keep our software free from malicious attacks like ransomware attacks, and other security threats. Hence we choose to trade off high performance of our system for security. There are other ways to increase the performance of the system. But this can't be negotiated with security.

## 5.7 Discussion of an alternate design

Skipped

# 6 Kubernetes experimentation (15%)

## 6.1 Experiment Design



CustomerConfig Microservice Experimentation

1. Experiment details:
    a. Web application: In this experiment, we used one of the many microservices of our application Intelli-Image. The microservice we have used is called CustomerConfig which fetches the details of the user account from AWS RDS database.
    b. Workflow of the experiment:
        i. When the client requests for account details, the request goes to the Classic Load Balancer service of AWS which is a network load balancer.
        ii. The load balancer directs the request to the kubernetes service of the Kubernetes cluster.
        iii. The kubernetes service redirects the request to the pod according to the criteria set for load balancing on pods.

    iv. The pod contains the docker container of our microservice CustomerConfig which processes the request by interacting with relational database AWS RDS.
    v. Once the request is processed, the response is sent back to the client.

c. Need of Kubernetes:
    i. When there is a huge number of requests for our microservice, use of just one instance is not sufficient. In order to fulfill all the requests, multiple instances of our microservice are necessary.
    ii. When the number of requests is small, use of multiple instances of it would be a waste of resources and money.
    iii. In order to solve this problem, the solution automatis that when there is a huge number of requests we should increase the number of instances of our microservice and when there is a less number of requests we should decrease the number of instances of our microservice.
    iv. The manual effort for this task is huge. So in order to address this problem we use Kubernetes which is an automated tool that provides a provision for scale up and scale down of the resources.

d. Use of Kubernetes in our experiment -
    i. Kubernetes provides the facility to autoscale the instances of our microservices.
    ii. In this, the horizontal pod autoscaler (HPA) is used which keeps track of the CPU and MEM usage by each pod.
    iii. We need to specify the criteria for the scale up and scale down of pods.
    iv. We need to specify the minimum pods possible for the service, maximum pods possible for the service and initial replica of the pods to start our service with.
    v. If the load exceeds the CPU and MEM mean utilization criteria specified then the Kubernetes autoscaler will increase the replicas of the pods accordingly.
    vi. If the load decreased than a threshold specified in the criteria then HPA will decrease the number of pods.

e. Handling of various load on our Microservice -
    i. We have used 2 Kubernetes nodes containing 2 pods initially.
    ii. We have set the criteria for the number of pods as 1 minimum pod, 6 maximum pods.
    iii. In our experiment, we have set the criteria for autoscale to be 60% mean CPU utilization.
    iv. If the mean CPU utilization is greater than 60% then the autoscaler spawns a new pod in our system in order to distribute the load.
    v. If the mean CPU utilization is very less then our autoscaler will decrease the number of pods one by one to monitor the load. If the load is very less so that it can be handled by one pod then the autoscaler will decrease the number of pods to the minimum number of replicas specified in the criteria.

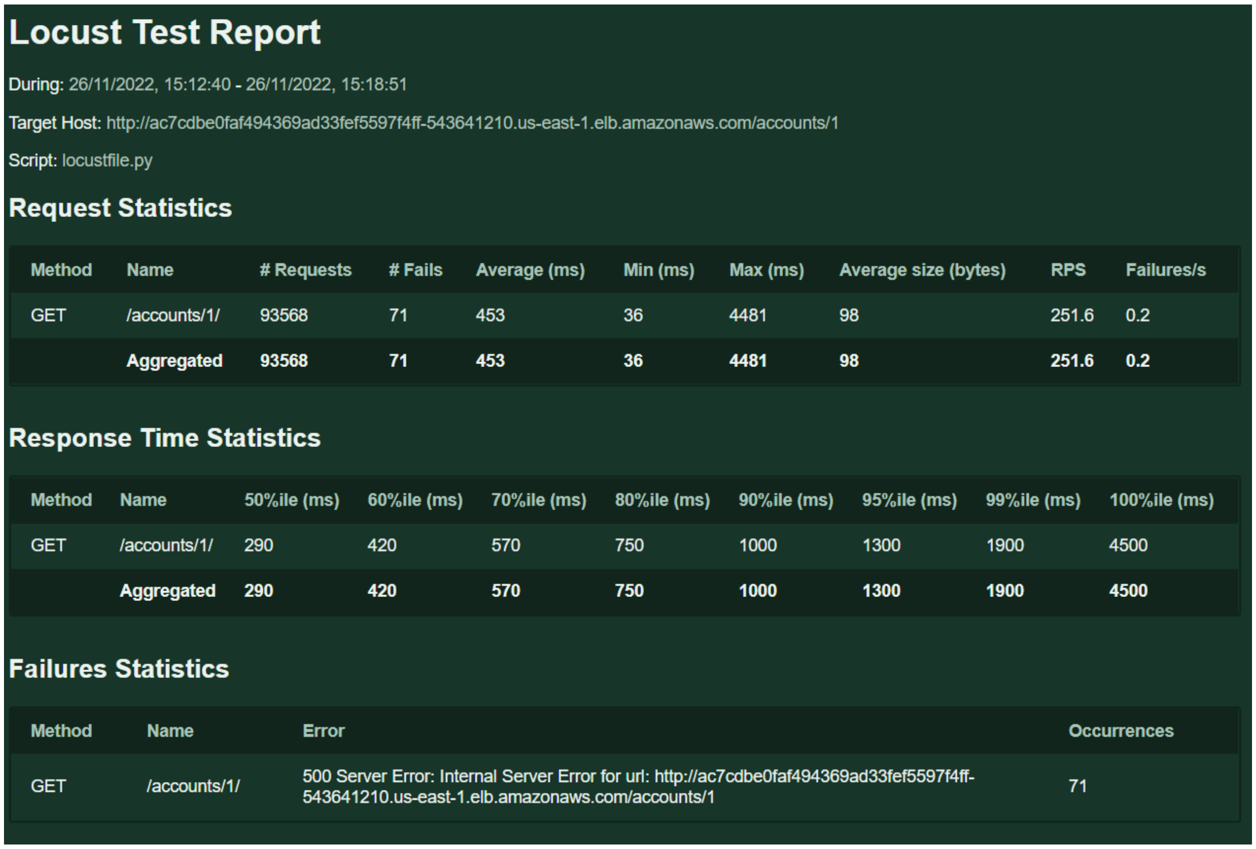
    f. Workload generation for testing -
        i. We have used a locust tool in order to generate a large amount of requests to test our Kubernetes autoscaler.

2. AWS and Kubernetes services used -
    a. Classic Load Balancer - The classic load balancer provides the functioning of distributing load across multiple EC2 instances.
    b. Elastic Compute Cloud (EC2) - It provides scalable computing capacity.
    c. Elastic Kubernetes Service/Elastic Container Kubernetes (EKS) - It gives the provision to run Kubernetes on AWS
    d. Elastic Container Registry (ECR) - This provides the facility to store docker images. Our docker image of the microservice is stored in ECR.
    e. Amazon Relational Database Service (RDS) - This service is used to store the relational data. We have stored the account details of the user in this database service.
    f. Kubernetes SVC- Kubernetes service is a logical abstraction for a deployed group of pods in a kubernetes cluster(All the pods have the same functionality)
    g. Kubernetes Node - Kubernetes node is a worker machine which is an EC2 instance on AWS.
    h. Kubernetes Pod - It is the single instance of our running microservice in the kubernetes cluster.
    i. Kubernetes HPA - Kubernetes Horizontal Pod Autoscaler is used to manage the number of pods in our cluster. It autoscales the pods according to the custom metrics.

## 6.2 Workload Generation with Locust

1. Usage of Locust tool -
   Locust is used for load testing of an application and to check whether the system can handle concurrent requests.
2. We have used locust to test our CustomerConfig microservice.
3. API to test - Get_Account_Details()
4. Target host (our microservice)- kubernetes service url - http://ac7cdbe0faf494369ad33fef5597f4ff-543641210.us-east-1.elb.amazonaws.com/accounts/1
5. Number of users (peak concurrency) - 1000
6. Spawn rate (users started/second) - 100
7. Total number of requests - 93568
8. Requests/second - 251 requests/second
9. Failure/sec - 0.2
10. Steps followed to run the locust test -
    a. Run the locustfile.py script which will start the locust server
    b. Add peak concurrency, spawn rate and target host url
    c. Run the test

11. Our request failure rate is very less so our autoscaler handled the load and spawned new pods using our docker image stored in ECR as per the requirement.
12. When there was no request, the autoscaler decreased the replicas to the minimum replicas as expected which is shown in the next section.
13. The locust test report details are mentioned here -



Locust Test Report

During: 26/11/2022, 15:12:40 - 26/11/2022, 15:18:51

Target Host: http://ac7cdbe0faf494369ad33fef5597f4ff-543641210.us-east-1.elb.amazonaws.com/accounts/1

Script: locustfile.py

Request Statistics

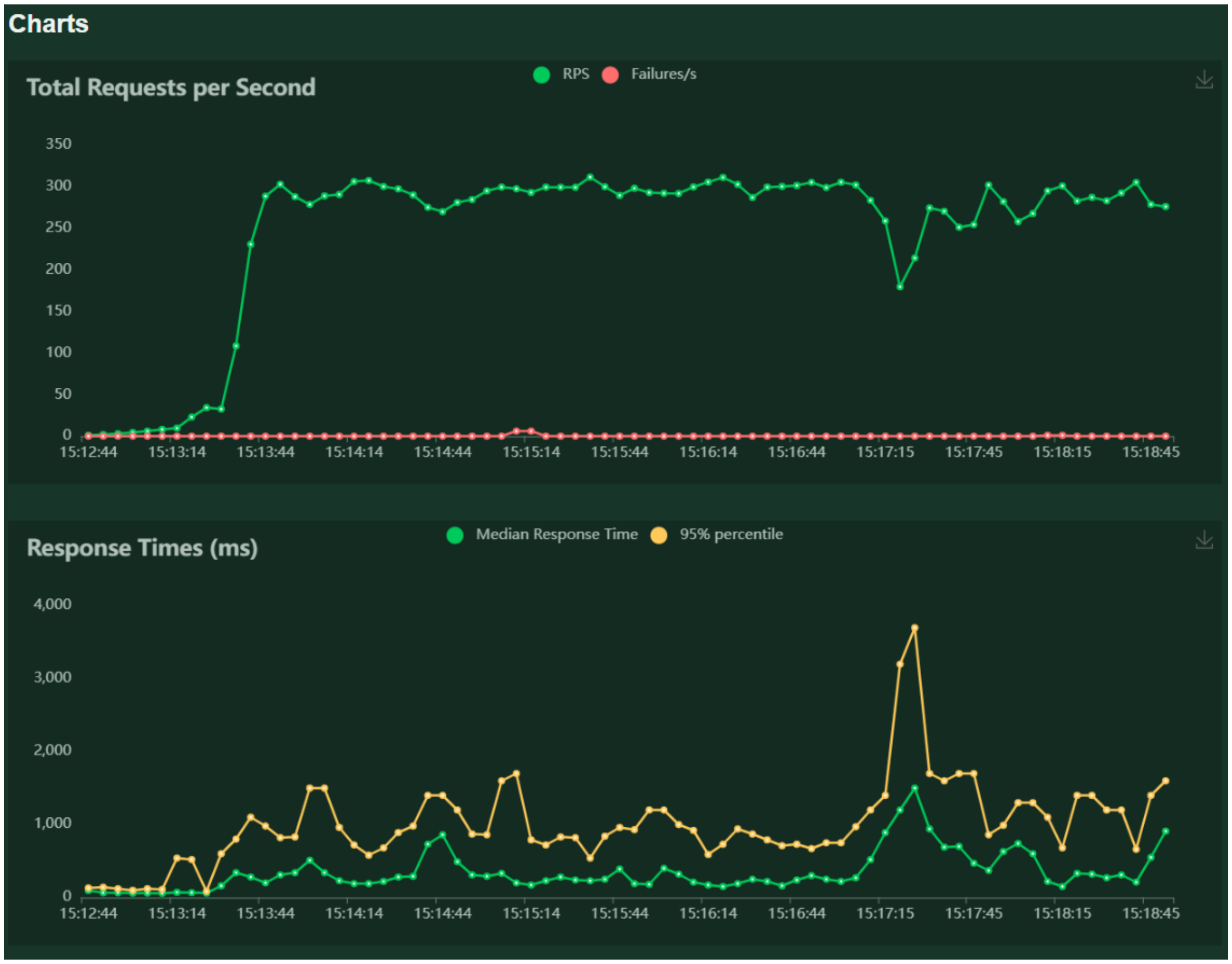
| Method | Name | # Requests | # Fails | Average (ms) | Min (ms) | Max (ms) | Average size (bytes) | RPS | Failures/s |
|---|---|---|---|---|---|---|---|---|---|
| GET | /accounts/1/ | 93568 | 71 | 453 | 36 | 4481 | 98 | 251.6 | 0.2 |
| | **Aggregated** | **93568** | **71** | **453** | **36** | **4481** | **98** | **251.6** | **0.2** |

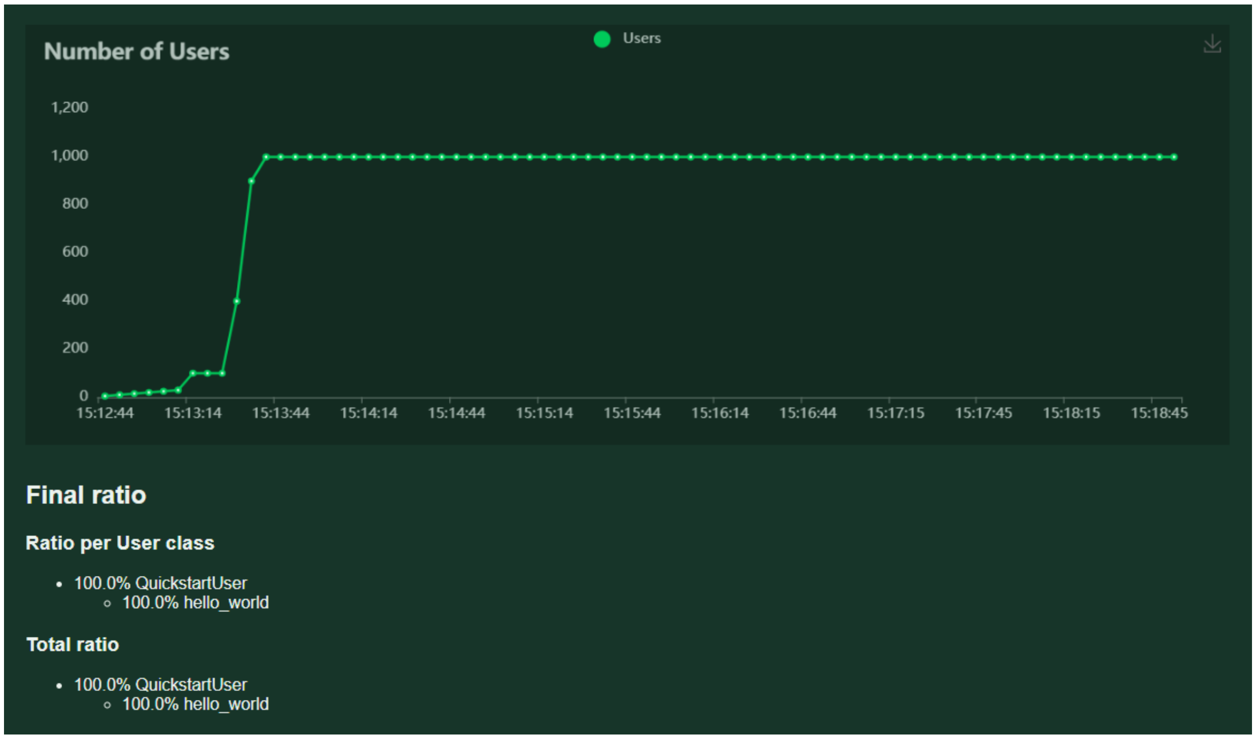
Response Time Statistics

| Method | Name | 50%ile (ms) | 60%ile (ms) | 70%ile (ms) | 80%ile (ms) | 90%ile (ms) | 95%ile (ms) | 99%ile (ms) | 100%ile (ms) |
|---|---|---|---|---|---|---|---|---|---|
| GET | /accounts/1/ | 290 | 420 | 570 | 750 | 1000 | 1300 | 1900 | 4500 |
| | **Aggregated** | **290** | **420** | **570** | **750** | **1000** | **1300** | **1900** | **4500** |

Failures Statistics

| Method | Name | Error | Occurrences |
|---|---|---|---|
| GET | /accounts/1/ | 500 Server Error: Internal Server Error for url: http://ac7cdbe0faf494369ad33fef5597f4ff-543641210.us-east-1.elb.amazonaws.com/accounts/1 | 71 |


Charts
Total Requests per Second
RPS
Failures/s
350
300
250
200
150
100
50
0
15:12:44
15:13:14
15:13:44
15:14:14
15:14:44
15:15:14
15:15:44
15:16:14
15:16:44
15:17:15
15:17:45
15:18:15
15:18:45
Response Times (ms)
Median Response Time
95% percentile
4,000
3,000
2,000
1,000
0
15:12:44
15:13:14
15:13:44
15:14:14
15:14:44
15:15:14
15:15:44
15:16:14
15:16:44
15:17:15
15:17:45
15:18:15
15:18:45



**Final ratio**

**Ratio per User class**

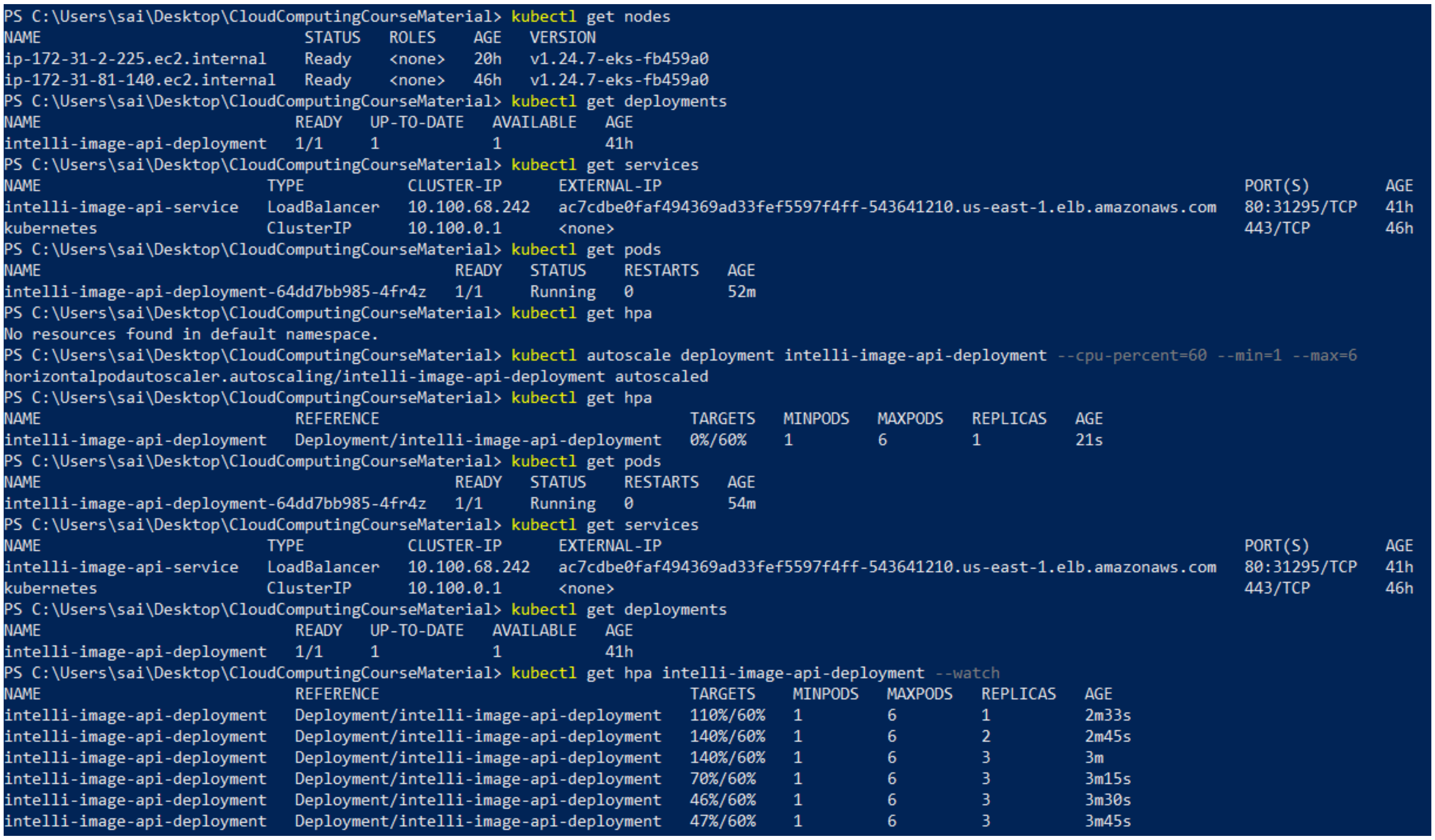
- 100.0% QuickstartUser
  - 100.0% hello_world

**Total ratio**

- 100.0% QuickstartUser
  - 100.0% hello_world

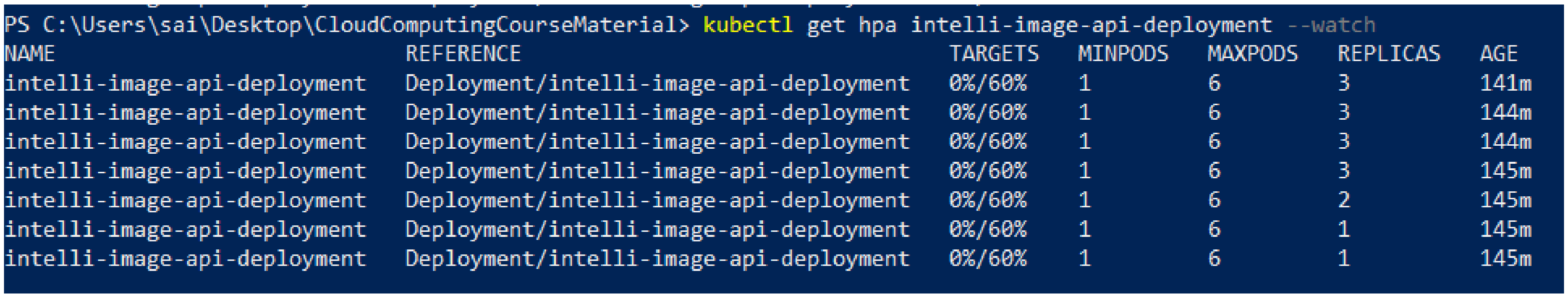
## 6.3 Analysis of the results

1. Initially we have 2 Kubernetes nodes and 1 pod running on our EKS service.
2. We add the autoscalar by setting CPU_percent=60 i.e. mean CPU utilization=60%, minimum pod=1 i.e. minimum number of pods our deployment should have, maximum pod=6 maximum number of pods our deployment can have.
3. Name of our deployment is - intelli-image-api-deployment
4. As our min replica count was 1 so in our deployment replica count initially was 1.
5. When we started the load testing using Locust tool, our load generated 1000 peak concurrency and 251 request/sec, so CPU utilization of our pod increased to around 110%.
6. In our autoscaler we had set the CPU utilization to 60% so it scaled up the number of pods in our deployment from 1 to 2.
7. But due to increased load our CPU utilization reached to 140% so it again scaled up the number of pods in our deployment from 2 to 3.
8. After spawning 2 more pods, the total number of pods was 3 and the mean CPU utilization was around 46% which is within the limit of our criteria so our services stabilized.



```
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get nodes
NAME                          STATUS   ROLES    AGE   VERSION
ip-172-31-2-225.ec2.internal    Ready    <none>   20h   v1.24.7-eks-fb459a0
ip-172-31-81-140.ec2.internal   Ready    <none>   46h   v1.24.7-eks-fb459a0
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get deployments
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
intelli-image-api-deployment   1/1     1            1           41h
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get services
NAME                       TYPE           CLUSTER-IP      EXTERNAL-IP                                                               PORT(S)        AGE
intelli-image-api-service   LoadBalancer   10.100.68.242   ac7cdbe0faf494369ad33fef5597f4ff-543641210.us-east-1.elb.amazonaws.com   80:31295/TCP   41h
kubernetes                 ClusterIP      10.100.0.1      <none>                                                                    443/TCP        46h
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get pods
NAME                                            READY   STATUS    RESTARTS   AGE
intelli-image-api-deployment-64dd7bb985-4fr4z   1/1     Running   0          52m
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get hpa
No resources found in default namespace.
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl autoscale deployment get intelli-image-api-deployment --cpu-percent=60 --min=1 --max=6
horizontalpodautoscaler.autoscaling/intelli-image-api-deployment autoscaled
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get hpa
NAME                          REFERENCE                                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         1          21s
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get pods
NAME                                            READY   STATUS    RESTARTS   AGE
intelli-image-api-deployment-64dd7bb985-4fr4z   1/1     Running   0          54m
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get services
NAME                       TYPE           CLUSTER-IP      EXTERNAL-IP                                                               PORT(S)        AGE
intelli-image-api-service   LoadBalancer   10.100.68.242   ac7cdbe0faf494369ad33fef5597f4ff-543641210.us-east-1.elb.amazonaws.com   80:31295/TCP   41h
kubernetes                 ClusterIP      10.100.0.1      <none>                                                                    443/TCP        46h
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get deployments
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
intelli-image-api-deployment   1/1     1            1           41h
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get hpa intelli-image-api-deployment --watch
NAME                          REFERENCE                                 TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   110%/60%   1         6         1          2m33s
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   140%/60%   1         6         2          2m45s
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   140%/60%   1         6         3          3m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   70%/60%    1         6         3          3m15s
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   46%/60%    1         6         3          3m30s
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   47%/60%    1         6         3          3m45s
```

9. When we stopped the load testing, the mean CPU utilization went to 0% so ideally we don't need 3 pods in our deployment. Hence autoscaler scaled down the number of pods to minimum number of pods i.e. 1



```
PS C:\Users\sai\Desktop\CloudComputingCourseMaterial> kubectl get hpa intelli-image-api-deployment --watch
NAME                          REFERENCE                                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         3          141m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         3          144m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         3          144m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         3          145m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         2          145m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         1          145m
intelli-image-api-deployment   Deployment/intelli-image-api-deployment   0%/60%    1         6         1          145m
```

10. Result -

| **Test scenario** | **Expected output** | **Actual output** |
|---|---|---|
| When the load is less (mean CPU utilization is < 60%) | Number of pods currently being used should be equal to the number of minimum pods required to keep the mean CPU utilization < 60% | • It is the same as expected output.<br>• When there was no load initially, the number of pods in our deployment was 1 |

| | | |
|---|---|---|
| | | which is the minimum number of pods required. |
| When the load is high (mean CPU utilization is > 60%) | Number of pods currently being used should increase till the mean CPU utilization becomes < 60% | ● It is the same as expected output.<br>● When load increases, the number of pods in our deployment increases to 3 after which the mean CPU utilization becomes within our threshold i.e. 60%. |

# 7 Ansible Playbooks

Skipped

## 8 Demonstration (0%)

We chose not to do a demonstration.

# 9 Comparisons

Skipped

# 10 Conclusion (5%)

## 10.1 Lessons Learned

We had the following thoughts as we reached the conclusion of this project:

- The process of identifying conflicting requirements was by far the most rewarding yet time-consuming exercise.
- The AWS WAF was a surprisingly excellent resource for coming up with requirements. Not only did it provide us with a lot of ready-made, tried and tested technical requirements which have been applicable to real life projects, it also gave us related examples, benefits, best practices and documentation. The WAF is definitely something we are going to use in the future, perhaps while wearing one of the “hats” described in the course.
- We faced a last-minute sprint to the deadline. This could have been avoided had there been stricter deadlines for section-wise approval from the professor. It would have placed slightly more pressure on us during the semester, but it would have made our life easier towards the end of the semester.

## 10.2 Possible continuation of the project

None of the group members plan to take the advanced course in cloud computing in spring 2023; therefore we will not be continuing this project.

# 11 References

**[1]**

AWS Well-Architected - Build secure, efficient cloud applications - https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc&wa-guidance-whitepapers.sort-by=item.additionalFields.sortDate&wa-guidance-whitepapers.sort-order=desc

**[2]**

Cloud Pricing Comparison - Cast AI - https://cast.ai/blog/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-platform/

**[3]**

Amazon VPC Features - https://aws.amazon.com/vpc/features/

**[4]**

Amazon DynamoDB Features | NoSQL Key-Value Database | Amazon Web Services. *AWS*, https://aws.amazon.com/dynamodb/features/

**[5]**

Application Load Balancer | Elastic Load Balancing | Amazon Web Services - https://aws.amazon.com/elasticloadbalancing/application-load-balancer/?nc=sn&loc=2&dn=2

**[6]**

Amazon S3 Security Features - Amazon Web Services - https://aws.amazon.com/s3/security/?nc=sn&loc=5

**[7]**

DynamoDB encryption at rest - Amazon DynamoDB - https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html

**[8]**

Data protection in Amazon EC2 - Amazon Elastic Compute Cloud - https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/data-protection.html#encryption-transit

**[9]**

Amazon Well Architected Framework - Performance Efficiency Whitepaper. *YouTube*, https://docs.aws.amazon.com/pdfs/wellarchitected/latest/performance-efficiency-pillar/wellarchitected-performance-efficiency-pillar.pdf#welcome.

**[10]**

Amazon Well Architected Framework - Operational Excellence Whitepaper https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wllarchitected-operational-excellence-pillar.pdf#welcome

**[11]**

Amazon Well Architected Framework - Reliability Whitepaper - https://docs.aws.amazon.com/pdfs/wellarchitected/latest/reliability-pillar/wellarchitected-reliability-pillar.pdf#welcome

**[12]**

SaaS Solutions on AWS - https://d1.awsstatic.com/whitepapers/saas-solutions-on-aws-final.pdf

**[13]**

Golding, Tod. “Modeling SaaS Tenant Profiles on AWS | AWS Partner Network (APN) Blog.” *Amazon AWS*, 8 March 2016, https://aws.amazon.com/blogs/apn/modeling-saas-tenant-profiles-on-aws/ Accessed 26 November 2022.

**[14]**

Amazon EC2 Auto Scaling benefits - Amazon EC2 Auto Scaling - https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-benefits.html#arch-AutoScalingMultiAZ

**[15]**

Attribute-Based Access Control – AWS Identity - https://aws.amazon.com/identity/attribute-based-access-control/

**[16]**

Security Pillar - AWS Well-Architected Framework - https://docs.aws.amazon.com/pdfs/wellarchitected/latest/security-pillar/wellarchitected-security-pillar.pdf#welcome

**[17]**

Operational Excellence Pillar - AWS Well-Architected Framework - https://docs.aws.amazon.com/pdfs/wellarchitected/latest/operational-excellence-pillar/wellarchitected-operational-excellence-pillar.pdf#welcome