CSC 547 Cloud Computing Architecture
Fall 2023

Dr. Yannis Viniotis and Ionnis Papapanagiotou

Project Name: Malware Threat Detection

Arohan Ajit (200538209)

1. Introduction

This SaaS application is designed to help businesses of all sizes automate and streamline their cybersecurity requirements by providing an automated malware detection mechanism. The application provides a fast and reliable way to scan files for possible malware threats.

1.2 Executive Summary

The Malware Threat Detection SaaS application is a cutting-edge cloud-based security solution, purpose-built to fulfill the critical need for robust malware detection and prevention in the digital age. This advanced application is hosted in the cloud, ensuring flexibility, scalability, and easy accessibility. Designed to tackle a wide spectrum of malware threats, including viruses, ransomware, spyware, and zero-day attacks, this cloud-native solution offers a seamless and hassle-free deployment experience. Real-time updates and continuous monitoring of network traffic and endpoints further enhance its cloud-powered capabilities, enabling proactive threat identification and containment.

1.3 Motivation

Cybersecurity is an ever evolving field. Traditional on premises solutions are lagging behind owing to constant updates required in order to keep up with threats developing each day. Cloud hosted malware threat detection solutions offer a centralized, scalable and cost effective way of countering these threats and securing assets. Cloud based security solutions have a key advantage of access to a large amount of data in order to identify and remove various patterns of malicious activity. These architectures provide real time protection against latest malwares by constantly monitoring files hosted on cloud. They have no issue scaling up or down as per workload and provide a cost effective solution to ensure the entire file system is secure and free from any malwares. Cloud systems are easy to deploy and are proactive in threat detection through automated updates and patches. Utilizing vast infrastructure hosted by cloud providers, the clients can create flexible security infrastructure which can have better performance and utilize better threat intelligence on prem infrastructure.

2. Problem Description

2.1 The Problem

The projects aims to achieve higher cyber security resilience in cloud native application through cloud based malware detection

2.2 Business Requirements:

BR-1 Accommodate time varying workloads
BR-2 Must have high performance
BR-3 Should have high availability
BR-4 Process data in real time
BR-5 Detect and prevent fraud
BR-6 Should be secure
BR-7 Adhere to global compliant standards
BR-8 Integrate with multiple tools and services
BR-9 Support multiple platforms and operating systems
BR-10 Must be able to support multiple instances of the service simultaneously (multi tenancy)
BR-11 Authenticate and validate users
BR-12 Cost optimization
BR-13 Have disaster recovery and backup mechanisms
BR-14 Incorporate ML algorithms for better detection accuracy
BR-15 Monitor logs and metrics for better analytics
BR-16 Define data and privacy policies to protect user data
BR-17 Have defined billing and subscription models
BR-18 Analyze and infer statistics based on logged data for improvement and training

2.3 Technical Requirements:

TR-1 Implement auto-scaling to accommodate time varying workloads
TR-2 Design the architecture to optimize resource allocation for high performance.
TR-3 Use redundancy, load balancing, and failover mechanisms to ensure high availability
TR-4 Deploy real-time data processing pipelines for immediate data analysis.
TR-5 Incorporate fraud detection algorithms and rules into the architecture
TR-6 Implement encryption, access controls, and authentication mechanisms to ensure security.
TR-7 Adhere to global compliance standards by implementing data protection and auditing features
TR-8 Provide APIs and connectors to integrate with multiple tools and services
TR-9 Ensure compatibility with various platforms and operating systems for multi-platform and operating systems support
TR-10 Support multi-tenancy by implementing isolation mechanisms for concurrent service

instances.

TR-11 Implement user authentication and validation mechanisms for secure access
TR-12 Utilize cost monitoring and resource optimization tools to control expenses.
TR-13 Design and test disaster recovery and backup mechanisms to ensure data availability.
TR-14 Integrate machine learning algorithms for improved fraud detection accuracy.
TR-15 Implement logging and monitoring systems for analytics and performance tracking.
TR-16 Define and enforce data and privacy policies to protect user data
TR-17 Implement billing and subscription models for service monetization. (Tenant Identification)
TR-18 Use data analytics to analyze and infer statistics from logged data for system improvement and training.

2.4 Tradeoffs:

Tradeoff 1: Auto Scaling vs Cost Optimization (TR-1 / TR-12)

- Prioritizing auto scaling is essential for handling increased workload and traffic variability in malware detection applications.
- Cost optimization might conflict with auto scaling as dynamically allocating resources to meet increased demand could lead to higher infrastructure costs.

Tradeoff 2: Redundancy, Load Balancing, and Failover vs Cost Optimization (TR-3 / TR-12)

- Implementing redundancy, load balancing, and failover mechanisms is crucial for continuous service availability and threat protection.
- Achieving high availability often requires redundancy, which may lead to increased infrastructure costs.

Tradeoff 3: Encryption, Access Controls, and Authentication vs Resource Allocation for High Performance (TR-2 / TR-6)

- Encryption, access controls, and authentication are essential for protecting sensitive data and ensuring regulatory compliance.
- Implementing strong security measures may consume resources and potentially impact system performance.

We get into more detail of these tradeoffs in Section 5.

3. Provider Selection

3.1 Criteria for choosing Provider

(In priority order starting from most to least priority)

1. Service offered
2. Security features available
3. Compliance certificates
4. Data management tools and data transfer speeds
5. Variety of data storage options available
6. Performance metrics - network speed
7. Reputation
8. Cost and pricing of various services
9. Track performance of uptime and reliability
10. Availability of data centers around the world
11. Variety of solutions available for scalability
12. Backup and Recovery metrics
13. Options available for Operation Systems, Programming languages, frameworks and databases
14. Customer Support Response times
15. Vendor lock in - data formats used, termination clauses in SLA
16. Roadmap - ensuring the provider uses latest up to date technologies

3.2 Provider Comparison

- AWS has the *most extensive* and *mature set* of services while providing a well-established and flexible pricing model. Azure also offers a comprehensive set of services while AWS has a more extensive portfolio. Azure excels in integration with Microsoft products and ecosystem. GCP has a robust set of services particularly excelling in data analytics and machine learning.
- Azure offers competitive pricing and discounts, especially for enterprises using Microsoft products but AWS's pricing flexibility is often considered superior. GCP is known for transparent and straightforward pricing, but AWS and Azure often provide more flexibility in pricing models.
- All three providers have a strong track record for uptime and reliability, but AWS's longer time in the market and larger global infrastructure contribute to its slightly higher reliability perception.
- Security is a top priority for all three providers, offering a wide range of tools for identity management, encryption, and compliance. However, AWS's longer tenure in the market has contributed to its reputation for robust security features.
- They have a comprehensive set of compliance certifications, ensuring adherence to industry and regulatory standards.
- They also offer scalable solutions with auto-scaling capabilities. AWS's extensive range of instance types and global infrastructure provides scalability advantages.
- All three support a wide variety of operating systems, programming languages, frameworks, and databases. AWS's extensive partner ecosystem and integration capabilities contribute to its leading position.

- Each provider has its ecosystem, potentially leading to some level of vendor lock-in. AWS's extensive partner network and integrations, however, provide more flexibility and options for mitigating lock-in concerns.
- All three providers regularly update services and infrastructure to incorporate the latest technologies.

Ref[1]

- **AWS vs Microsoft Azure vs Google GCP (comparison of different metrics):**

Detail	Amazon AWS	Microsoft Azure	Google GCP
Minimum Instance	2 virtual CPUs, and 8 GB of Ram will price you around – USD 69/month	2 virtual CPUs, and 8 GB of Ram will price you around – USD 70/month	2 virtual CPUs, and 8 GB of Ram will price you around – USD 52/month
Maximum Instance	3.84 TB Ram, 128 vCPUs will price you around – USD 3.97/hour	3.89 TB Ram, 128 v CPUs will price you around – USD 6.97/hour	3.75 TB Ram, 160 v CPUs will price you around – USD 5.32/hour
Type of Discount	Reserved Instances (RIs)	Reserved Instances (RIs)	Committed Use Discount (CUD)

			Sustained Use Discount (SUD)
Commitment	1 or 3 years	1 or 3 years	Committed Use Discount (CUD) – 1 or 3 years Sustained Use Discount (SUD) – no commitment
Discount percentage	Up to 75 percent	Up to 72 percent	Committed Use Discount (CUD) – for 1 year up to 37 percent or 3 years up to 55 percent Sustained Use Discount (SUD) – up to 30 percent
Is cancellation available?	Yes, it offers to sell your products on the marketplace	Yes, they will charge a 12% cancellation fee	No cancellation is available
Payment options	3 options are available on AWS – no up-front, partial up-front, all up-front	All up-front	No up-front
High Profile Customers	LinkedIn, Facebook, BBC, Airbnb, Twitch, Netflix, Adobe, ESPN, Lamborghini, etc.	Apple, HP, Coca-Cola, LG Electronics,	Twitter, Intel, Yahoo, PayPal, eBay, Target, 20th Century Fox, etc.Cessation

Ref[2]

- Based on the table above, AWS is clearly outranking Azure and GCP in terms of cost, some aspects of reliability and reputation.
- The biggest difference between the three services is the service offered for secure file transfer.
- Most AWS customers expect somewhere between 99.9% to 99.999% uptime, which translates to roughly 53 minutes to 8.76 hours per year of allowable outage time

Ref [3]

Backup and Disaster Recovery Metrics

	AWS Elastic Disaster Recovery	Azure Site Recovery	Google Cloud Backup and DR
Portfolio	✅		
Partners	✅		
Use Cases	✅		
Ratings		✅	
Recognition	✅		
Overall	✅		

Cloud Security Comparison

Metrics	AWS	Azure	GCP
Shared Responsibility Model	Divided into 2 categories	Divided into 3 categories which are loosely correlated	A complex bifurcation of all the metrics for the model
IAM	No built in IAM service	Azure Privileged Identity Management	No built in IAM service
VPN	10 site to site connection limit with point to site option	30 site to site connection limit with point to site option	No point to site option available
Marketplace support for security options	Best	Best	Medium

Ref[4]

- We can observe through these metrics that AWS best meets our criteria, after that is Azure and the least one is GCP.

3.3 The final selection

- Based on the comparisons in section 3.2, clearly AWS is the winner.

3.3.1 The list of services offered by the winner

- https://aws.amazon.com/products/

Services corresponding to the TRs:

TR-1 Implement auto-scaling to accommodate time varying workloads
Services: AWS CodeBuild, AWS Transfer Family server and AWS Lambda function.

TR-2 Design the architecture to optimize resource allocation for high performance.
Services: AWS Transfer Family, Amazon S3, AWS Lambda, and Amazon ECR are components that contribute to resource allocation and performance.

TR-3 Use redundancy, load balancing, and failover mechanisms to ensure high availability
Services: Amazon S3 for redundant storage, AWS Transfer Family with possible Auto Scaling for load balancing, and AWS Lambda for failover.

TR-4 Deploy real-time data processing pipelines for immediate data analysis.
Services: Amazon Kinesis for real-time data streaming and processing

TR-5 Incorporate fraud detection algorithms and rules into the architecture
Services: AWS Service in Diagram: AWS Lambda for running the ClamAV (fraud detection) algorithm.

TR-6 Implement encryption, access controls, and authentication mechanisms to ensure Security.
Services: AWS Transfer Family for secure file transfers, Amazon S3 for storage encryption, AWS Lambda for secure code execution

TR-7 Adhere to global compliance standards by implementing data protection and auditing Features
Services: AWS CloudTrail for compliance and auditing.

TR-8 Provide APIs and connectors to integrate with multiple tools and services

Services: AWS Transfer Family for file transfer and Amazon API Gateway for API integration.

TR-9 Ensure compatibility with various platforms and operating systems for multi-platform and operating systems support
Services: Amazon S3 and AWS Lambda supporting diverse platforms and operating systems.

TR-10 Support multi-tenancy by implementing isolation mechanisms for concurrent service Instances.
Services: AWS Transfer Family with AWS Lambda and Amazon S3 for isolation mechanisms.

TR-11 Implement user authentication and validation mechanisms for secure access
Services: AWS Lambda and AWS Secrets Manager

TR-12 Utilize cost monitoring and resource optimization tools to control expenses.
Services: AWS Cost Explorer

TR-13 Design and test disaster recovery and backup mechanisms to ensure data availability.
Services: AWS Disaster Recovery for backup and disaster recovery planning

TR-14 Integrate machine learning algorithms for improved fraud detection accuracy.
Services: Amazon SageMaker for machine learning.

TR-15 Implement logging and monitoring systems for analytics and performance tracking. (Mandatory TR)
Services: Amazon CloudWatch for logging and monitoring.

TR-16 Define and enforce data and privacy policies to protect user data
Services: AWS Lambda and AWS Secrets Manager for credential storage and authentication

TR-17 Implement billing and subscription models for service monetization.
Services: Handled in house by AWS

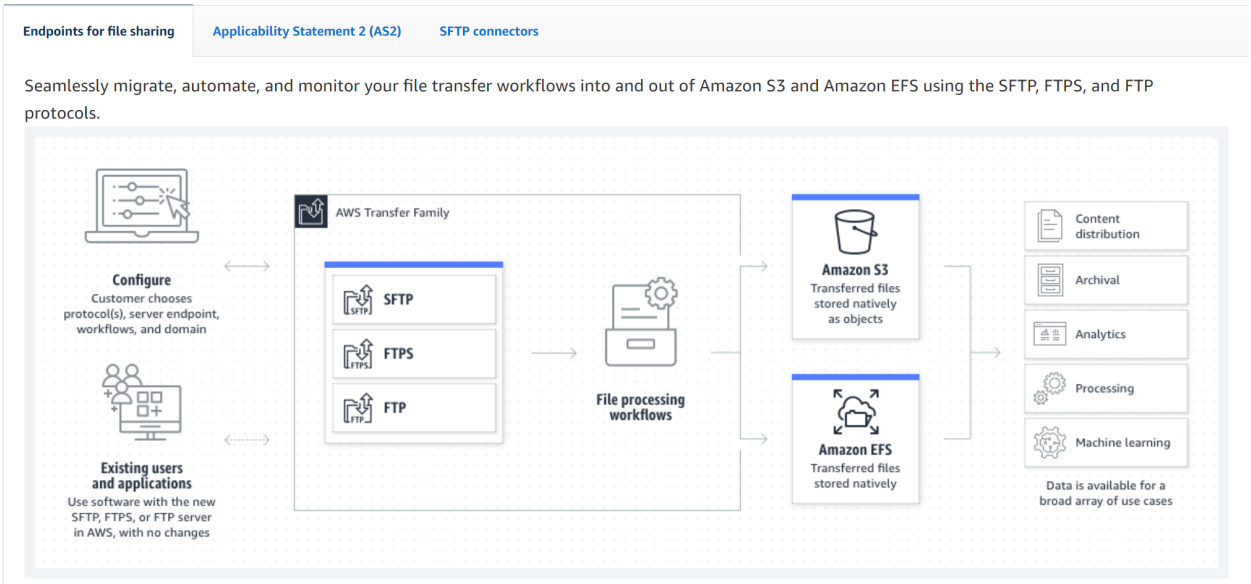
TR-18 Use data analytics to analyze and infer statistics from logged data for system improvement and training.
Services: Amazon QuickSight for data analytics

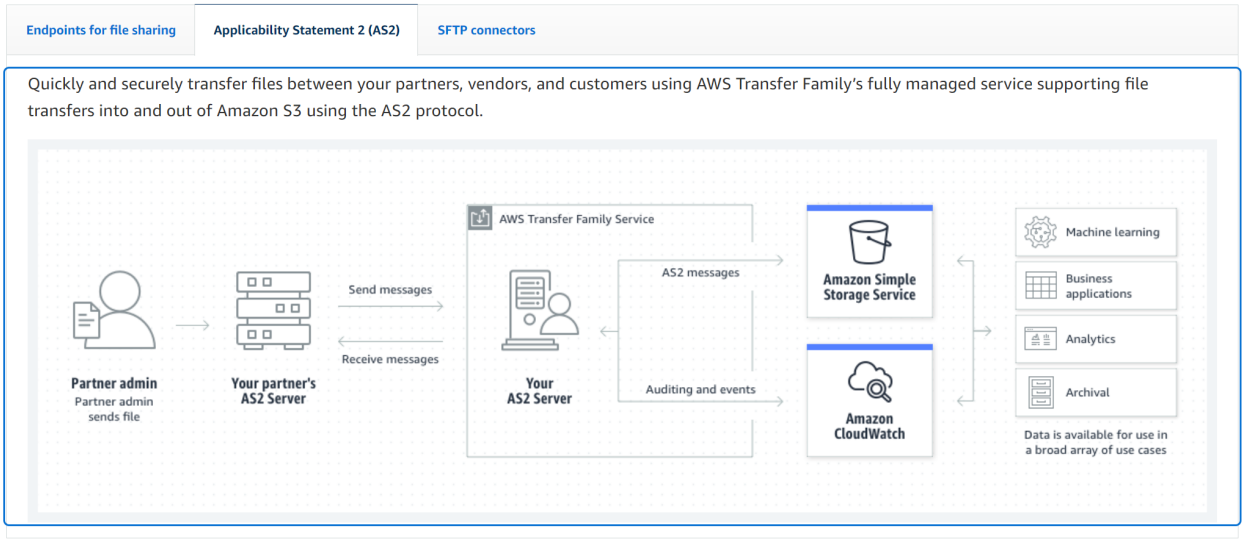
Some of the services are explained as below:

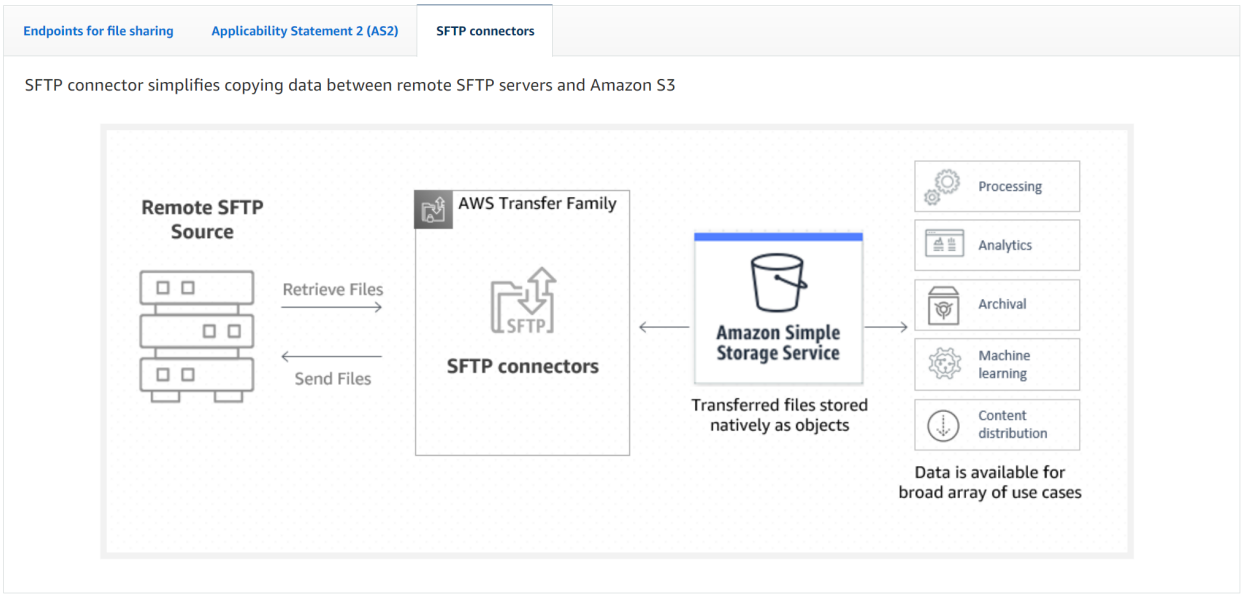
1. AWS Transfer Family: *Ref[5]*

- AWS Transfer Family is a secure transfer service that enables you to transfer files into and out of AWS storage services.

- Transfer Family provides the following benefits:
    - A fully managed service that scales in real time to meet your needs.
    - You don't need to modify your applications or run any file transfer protocol infrastructure.
    - With your data in durable Amazon S3 storage, you can use native AWS services for processing, analytics, reporting, auditing, and archival functions.
    - With Amazon EFS as your data store, you get a fully managed elastic file system for use with AWS Cloud services and on-premises resources. Amazon EFS is built to scale on demand to petabytes without disrupting applications, growing and shrinking automatically as you add and remove files. This helps eliminate the need to provision and manage capacity to accommodate growth.
    - A fully managed, serverless File Transfer Workflow service that makes it easy to set up, run, automate, and monitor processing of files uploaded using AWS Transfer Family.
    - There are no upfront costs, and you pay only for the use of the service.







Use Cases:

- Modernize your managed file transfers
    - Modernizing secure file transfers with financial and healthcare institutions for regulated data like PCI, PII, or HIPAA.
- Gain insights by growing your data lake
    - Enabling insights by connecting common business applications and IoT devices to a data lake for analysis and data processing.
- Improve collaboration across your trading partner network
    - Increasing collaboration and connectivity across supply chain trading partners to drive real-time insights across business applications like ERP, Transportation Management Systems, or other systems.
- Expand your content distribution business

- Expand your subscriber reach with multiple connectivity options. Apply built-in, fine-grained access controls to protect your revenue.

2. Amazon S3: *Ref[6]*

- Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

Use Cases:

- Build a data lake
  - Run big data analytics, artificial intelligence (AI), machine learning (ML), and high performance computing (HPC) applications to unlock data insights.

- Backup and restore critical data
  - Meet Recovery Time Objectives (RTO), Recovery Point Objectives (RPO), and compliance requirements with S3's robust replication features.

- Archive data at the lowest cost
  - Move data archives to the Amazon S3 Glacier storage classes to lower costs, eliminate operational complexities, and gain new insights.

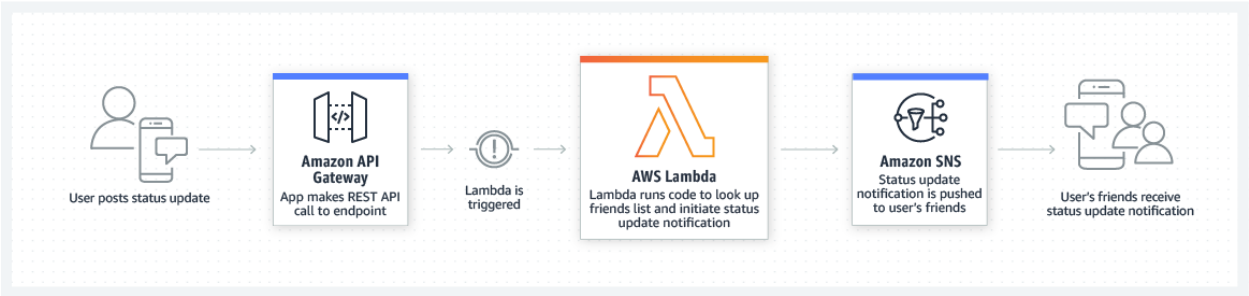
- Run cloud-native applications

- Build fast, powerful mobile and web-based cloud-native apps that scale automatically in a highly available configuration.

3. AWS Lambda: *Ref[7]*

- AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. You can trigger Lambda from over 200 AWS services and software as a service (SaaS) applications, and only pay for what you use.

AWS Lambda being used for Mobile backends



- Features
  - AWS Lambda allows you to add custom logic to AWS resources such as Amazon S3 buckets and Amazon DynamoDB tables, so you can easily apply compute to data as it enters or moves through the cloud.
  - You can use AWS Lambda to create new backend application services triggered on demand using the Lambda application programming interface (API) or custom API endpoints built using Amazon API Gateway.
  - With AWS Lambda, there are no new languages, tools, or frameworks to learn. You can use any third- party library, even native ones. You can also package any code (frameworks, SDKs, libraries, and more) as a Lambda Layer, and manage and share them easily across multiple functions.
  - AWS Lambda manages all the infrastructure to run your code on highly available, fault tolerant infrastructure, freeing you to focus on building differentiated backend services. With Lambda, you never have to update the underlying operating system (OS) when a patch is released, or worry about resizing or adding new servers as your usage grows.
  - AWS Lambda maintains compute capacity across multiple Availability Zones (AZs) in each AWS Region to help protect your code against individual machine or data center facility failures.

    - AWS Lambda supports function packaging and deployment as container images, making it easy for customers to build Lambda-based applications using familiar container image tooling, workflows, and dependencies. Customers also benefit from Lambda's operational simplicity, automatic scaling with sub-second startup times, high availability, pay-for-use billing model, and native integrations with over 200 AWS services and software-as-a-service (SaaS) applications.
    - AWS Lambda invokes your code only when needed, and automatically scales to support the rate of incoming requests without any manual configuration. There is no limit to the number of requests your code can handle.
    - Use Amazon RDS Proxy to take advantage of fully managed connection pools for relational databases. RDS Proxy efficiently manages thousands of concurrent connections to relational databases, making it easy to build highly scalable, secure Lambda-based serverless applications interacting with relational databases.
    - Provisioned Concurrency gives you greater control over your serverless application performance. When turned on, Provisioned Concurrency keeps functions initialized and hyper-ready to respond in double-digit milliseconds.
- Use Cases
    - Quickly process data at scale - Meet resource-intensive and unpredictable demand by using AWS Lambda to instantly scale out to more than 18k vCPUs. Build processing workflows quickly and easily with a suite of other serverless offerings and event triggers.
    - Run interactive web and mobile backends - Combine AWS Lambda with other AWS services to create secure, stable, and scalable online experiences.
    - Enable powerful ML insights - Preprocess data before feeding it to your machine learning (ML) model. With Amazon Elastic File System (EFS) access, AWS Lambda handles infrastructure management and provisioning to simplify scaling.
    - Create event-driven applications - Build event-driven functions for easy communication between decoupled services. Reduce costs by running applications during times of peak demand without crashing or over-provisioning resources.

4. AWS Secrets Manager: *Ref[8]*

AWS Secrets Manager helps you manage, retrieve, and rotate database credentials, application credentials, OAuth tokens, API keys, and other secrets throughout their life-cycles. Many AWS services store and use secrets in Secrets Manager.

Secrets Manager helps you improve your security posture, because you no longer need hard-coded credentials in application source code. Storing the credentials in Secrets Manager helps avoid possible compromise by anyone who can inspect your application or the components. You replace hard-coded credentials with a runtime call to the Secrets Manager service to retrieve credentials dynamically when you need them.

With Secrets Manager, you can configure an automatic rotation schedule for your secrets. This enables you to replace long-term secrets with short-term ones, significantly reducing the risk of compromise. Since the credentials are no longer stored with the application, rotating credentials no longer requires updating your applications and deploying changes to application clients.

Use Cases:

- Move hard-coded secrets to AWS Secrets Manager
- Move hard-coded database credentials to AWS Secrets Manager
- Set up alternating users rotation for AWS Secrets Manager
- Set up single user rotation for AWS Secrets Manager
- Create and manage secrets
- Control access to your secrets
- Retrieve secrets
- Rotate secrets
- Monitor secrets
- Audit secrets for compliance
- Create secrets in AWS CloudFormation

4. The first design draft

4.1 The basic building blocks of the design

- AWS Transfer Family Server:

  Ref[5]
  The AWS Transfer Family server is used for secure file transfers. It supports various protocols like FTP, FTPS, and SFTP, making it a versatile solution for transferring files securely.

- Amazon S3:

  Ref[6]
  Amazon S3 is a highly durable and scalable object storage service. It is utilized in the architecture to provide redundant storage for files. Files uploaded to the Transfer Family server are stored in S3 buckets, ensuring data durability and availability.

- AWS Lambda:

  Ref[7]
  AWS Lambda is a serverless compute service that runs code in response to events. In this architecture, Lambda functions are used as part of the managed workflow to scan each uploaded file with a (fraud detection) algorithm. The results of the scan determine whether the file is infected or not.

- Amazon EventBridge:

  Ref[9]
  Amazon EventBridge is a serverless event bus service that makes it easy to connect different applications using events. In this architecture, an EventBridge Scheduler rule is configured with a cron expression to trigger events.

- AWS CodeBuild:

  Ref[10]
  AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages.

- Amazon ECR (Elastic Container Registry):

  Ref[11]

Amazon ECR is a fully managed container registry that makes it easy for developers to store, manage, and deploy Docker container images.

- Amazon Kinesis:

  Ref[12]
  Amazon Kinesis is a platform for streaming data on AWS. It is used in this architecture for real-time data streaming and processing. The exact details of how Kinesis is employed depend on the specific requirements of the real-time data processing pipelines.

- AWS CloudTrail:

  Ref[13]
  AWS CloudTrail records AWS API calls for auditing purposes. Together, they help ensure adherence to global compliance standards by implementing data protection and auditing features.

- Amazon SageMaker:

  Ref[14]
  Amazon SageMaker is a fully managed service that enables developers to build, train, and deploy machine learning models at scale. In this architecture, it is utilized to integrate machine learning algorithms for improved fraud detection accuracy.

- Amazon CloudWatch:

  Ref[15]
  Amazon CloudWatch is used for logging and monitoring systems. It provides a comprehensive set of tools for collecting, analyzing, and visualizing logs and metrics, facilitating analytics and performance tracking.

- Amazon QuickSight:

  Ref[16]
  These services are used for data analytics, allowing the analysis and inference of statistics from logged data for system improvement and training.

4.2 Top-level, informal validation of the design

The services mentioned in Section 4.1 can be justified on the basis of the technical requirements:
Ref[5-16], Ref[1]

TR-1: Implement auto-scaling to accommodate time-varying workloads

- AWS Auto Scaling dynamically adjusts the capacity of the Transfer Family server and Lambda function based on the workload. This ensures the system can handle varying workloads efficiently.

TR-2: Design the architecture to optimize resource allocation for high performance

- These services collectively optimize resource allocation:
    - Amazon S3 provides scalable and durable storage.
    - AWS Lambda enables serverless execution, allocating resources as needed.
    - Amazon ECR stores container images efficiently.

TR-3: Use redundancy, load balancing, and failover mechanisms to ensure high availability

- Amazon S3 provides redundant storage.
- AWS Transfer Family, potentially with Auto Scaling, supports load balancing.
- AWS Lambda functions can act as failover mechanisms in the workflow.

TR-4: Deploy real-time data processing pipelines for immediate data analysis

- Amazon Kinesis facilitates the creation of real-time data processing pipelines, allowing immediate analysis of incoming data.

TR-5: Incorporate fraud detection algorithms and rules into the architecture

- AWS Lambda executes the fraud detection algorithm on uploaded files, tagging them as based on the scan results.

TR-6: Implement encryption, access controls, and authentication mechanisms to ensure security

- AWS Transfer Family and Amazon S3 support secure file transfers and storage.
- AWS Lambda ensures secure code execution.

TR-7: Adhere to global compliance standards by implementing data protection and auditing features

- AWS CloudTrail records API calls, aiding in compliance and auditing.

TR-8: Provide APIs and connectors to integrate with multiple tools and services

- AWS Transfer Family is used for file transfer.
- Amazon API Gateway facilitates API integration with external tools and services.

TR-9: Ensure compatibility with various platforms and operating systems for multi-platform and operating systems support

- Amazon S3 and AWS Lambda support diverse platforms and operating systems.

TR-10: Support multi-tenancy by implementing isolation mechanisms for concurrent service instances

- AWS Transfer Family, AWS Lambda, AWS Secrets Manager and Amazon S3 contribute to the implementation of isolation mechanisms for concurrent service instances.

TR-11: Implement user authentication and validation mechanisms for secure access

- AWS Lambda and AWS Secrets Manager create a custom authentication mechanism for secure access

TR-12: Utilize cost monitoring and resource optimization tools to control expenses

- Handled in house by Amazon

TR-13: Design and test disaster recovery and backup mechanisms to ensure data availability

- AWS Backup provides automated backup solutions, and AWS Disaster Recovery can be used for disaster recovery planning.

TR-14: Integrate machine learning algorithms for improved fraud detection accuracy

- Amazon SageMaker is used for integrating machine learning algorithms to enhance fraud detection accuracy.

TR-15: Implement logging and monitoring systems for analytics and performance tracking

- Amazon CloudWatch is employed for logging and monitoring, providing analytics and performance tracking.

TR-16: Define and enforce data and privacy policies to protect user data

- AWS Secrets Manager is used for data encryption.
- IAM enforces access control.

TR-17: Implement billing and subscription models for service monetization

- AWS has a built in billing and subscription model implemented for all cloud systems hosted.

TR-18: Use data analytics to analyze and infer statistics from logged data for system improvement and training

- Amazon Quicksight and AWS Sagemaker enable data analytics for analyzing and inferring statistics from logged data, contributing to system improvement and training.

Section 5: The Second Design

5.1 Use of Well Architected Framework

Following are the distinct steps suggested by AWS Well Architected Framework

- The application should successfully support development and workloads, acquire insight into their operations, and continuously enhance supporting processes and procedures in order to generate business value.

- The application should use cloud technologies to safeguard data, systems, and assets in order to improve your security posture.
- The application should perform its intended function correctly and consistently when expected. This involves being able to run and test the workload during its entire lifecycle.
- The application should efficiently use computing resources to meet system needs, and to retain that efficiency as demand changes and technology improve.
- The application should run systems in order to provide commercial value at the lowest possible cost.
- The application should continuously enhance sustainability impacts by reducing energy consumption and enhancing efficiency across all workload components by maximizing the advantages of given resources while minimizing overall resources required.

5.2 Discussion of Pillars

Security

Ref[17]

Cloud security is crucial for protecting data, applications, and infrastructure in the cloud environment. Seven essential practices for cloud security include:

- Security Foundations: Establishing a strong foundation for cloud security involves defining clear security policies and procedures, implementing threat modeling, and adopting a least privilege access model.
- Identity and Access Management (IAM): IAM controls who can access cloud resources and what actions they can perform. It involves user authentication, authorization, and access control mechanisms.
- Detection: Continuously monitoring and detecting security threats is essential for identifying and responding to potential attacks promptly. This includes using security analytics, log monitoring, and vulnerability scanning tools.
- Infrastructure Protection: Protecting the underlying cloud infrastructure involves securing virtual machines, containers, and networks. This includes implementing firewalls, intrusion detection systems, and network segmentation strategies.
- Data Protection: Maintaining data confidentiality, integrity, and availability is critical for cloud security. This includes encryption, data loss prevention, and data backup and recovery solutions.
- Incident Response: Having a well-defined incident response plan is essential for effectively responding to security breaches. This plan should outline clear roles, responsibilities, and communication protocols.
- Application Security: Securing cloud-based applications involves protecting against vulnerabilities, injection attacks, and other application-level threats. This includes using web application firewalls, code reviews, and penetration testing.

Design Principles

To enhance workload security in the cloud, consider implementing these key principles:

- Establish a Strong Identity Foundation
- Maintain Traceability
- Apply Security at All Layers
- Automate Security Best Practices
- Protect Data in Transit and at Rest
- Minimize Direct Data Access
- Prepare for Security Events

Best Practices:

- Security - Ensuring secure workload operation requires consistent application of overarching security best practices across all domains
- IAM - Defining accounts, users, roles, and services that can perform actions in your AWS account. Building out granular policies that assign permissions to principals, aligned with their roles and responsibilities. Implementing strong password practices, enforcing multi-factor authentication (MFA), and avoiding credential sharing.
- Detective - Detective controls are essential security measures that help organizations identify and respond to potential security threats or incidents.
- Infrastructure Protection - Infrastructure protection encompasses control methodologies, such as defense in depth, necessary to meet best practices and organizational or regulatory obligations.
- Data Protection - data classification provides a way to categorize organizational data based on levels of sensitivity, and encryption protects data by way of rendering it unintelligible to unauthorized access.
- Incident Response - Putting in place the tools and access ahead of a security incident, then routinely practicing incident response through game days, will help you verify that your architecture can accommodate timely investigation and recovery.
- Application Security - Application security (AppSec) encompasses designing, building, and testing the security of your workloads.

Reliability

Ref[18]

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.

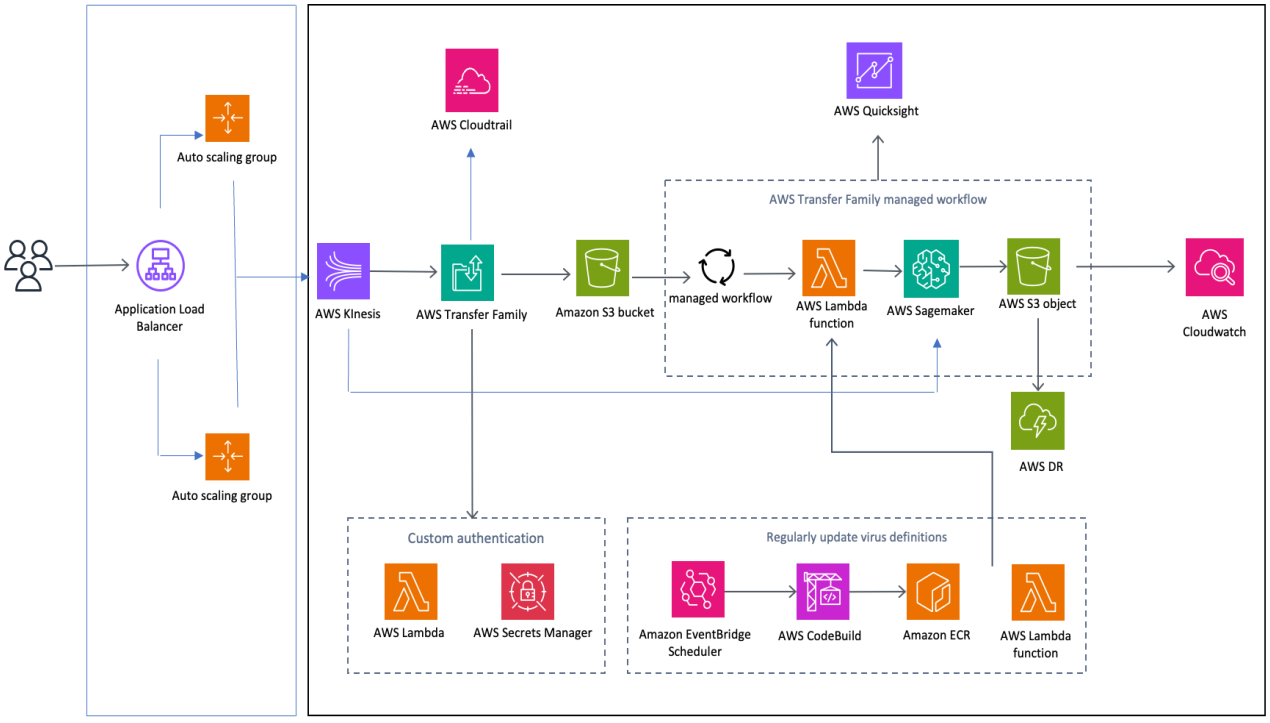
Design Principles

- Automatically recover from failure
- Test recovery procedures
- Scale horizontally to increase aggregate workload availability
- Stop guessing capacity
- Manage change in automation

Best Practices

- Foundations - For cloud-based workload architectures, there are Service Quotas (which are also referred to as service limits). These quotas exist to prevent accidentally provisioning more resources than you need and to limit request rates on API operations to protect services from abuse. Workloads often exist in multiple environments. You must monitor and manage these quotas for all workload environments.
- Workload - A reliable workload starts with upfront design decisions for both software and infrastructure. Your architecture choices will impact your workload behavior across all of the Well-Architected pillars. For reliability, there are specific patterns you must follow.
- Change Management - Changes to your workload or its environment must be anticipated and accommodated to achieve reliable operation of the workload. Changes include those imposed on your workload, such as spikes in demand, and also those from within, such as feature deployments and security patches.
- Failure Management - In any system of reasonable complexity, it is expected that failures will occur. Reliability requires that your workload be aware of failures as they occur and take action to avoid impact on availability. Workloads must be able to both withstand failures and automatically repair issues.

5.3 Use of Cloudformation Diagrams



Explanation:

- The application takes HTTP requests from the client system to connect to the remote cloud server. The application load balancer is used to distribute the load to two or more autoscaling groups. The auto scaling groups spread across different regions and availability zones in order to reduce latency, improve connection speed and to accommodate time varying workloads. In the autoscaling groups, a container orchestration service (for example Kubernetes) will host the interface that the users can interact with in order to use the application.
- The requests subsequently are sent to the remote server, where they first pass through AWS Kinesis for real time data collection and processing and then to the AWS Transfer family.
- The initial request to the Transfer Family server is passed on to custom authentication lambda function which utilizes AWS Secrets Manager for token identification and verification.
- Post successful authentication, the files are uploaded to AWS S3 Bucket via FTP using the AWS Transfer family. FIles are taken from the bucket and are scanned for malwares using Lambda function. The data from the scans are logged using Amazon Quicksight, and the same is also sent to SageMaker for further processing in the Machine Learning model. Post labeling, the files are stored in a separate S3 bucket. This bucket is also connected to AWS DR for appropriate backup and failover mechanisms if required.
- We use AWS Eventbridge scheduler, codebuild to create containers which can update virus definitions via internet and the data collected from scans on a periodic basis

5.4 Validation of Design

- Application Load Balancer and Autoscaling groups - Application load balancer are particularly well-suited for containerized environments, such as those using Docker and Kubernetes. They can route traffic to different containers based on rules and provide dynamic scaling for containerized workloads which we are using in the autoscaling groups portion. ALBs perform regular health checks on backend servers to ensure that they are responsive and capable of handling requests. Unhealthy servers are automatically removed from the rotation until they become healthy again, ensuring that only healthy servers receive traffic. As per the Explanation in section 5.3, the application load balancer and autoscaling groups fulfill **TR-1 Implement auto-scaling to accommodate time varying workloads** and **TR-2 Design the architecture to optimize resource allocation for high performance**.
  Trade Off - While this puts on a performance overhead, we have decided to prioritize high availability so that the application can better scale up and down in case there's an outbreak of a new malware for **trade off 1**
- AWS Kinesis - AWS Kinesis is a real time data processing service provided by AWS. This is used for real time data analysis for the files uploaded and helps calculate metrics for the clients on an immediate basis. This data can be provided to the machine learning model for processing at a later point in time. This service helps fulfill **TR-4 - Real time data processing for immediate data analysis**.

- AWS Cloudtrail - AWS Cloudtrail service is used for logging and monitoring application metrics for auditing purposes. Since the application deals with client data for malware detection, it'll need to adhere to compliance certificates and audits. This service plays a key role in logging data for the same and fulfilling **TR-7: Adhere to global compliance standards by implementing data protection and auditing features**
- AWS Transfer Family - AWS Transfer Family is the most important service to be used in this application. This service facilitates client authentication as well as secure file transfer from client system to the remote server. Amazon provides inbuilt auto scaling functionality with this service in order to automatically scale the load as per the files being transferred and implementing any failover mechanisms necessary. This fulfills **TR-1: Implement auto-scaling to accommodate time-varying workloads** and **TR-3: Use redundancy, load balancing, and failover mechanisms to ensure high availability.** Moreover, Transfer Family is highly versatile, with enabling authentication mechanism on its server. Our application provides a mechanism for custom user authentication where Transfer Family takes a request from the client system and sends it to a custom authentication function, thus helping fulfill **TR-6: Implement encryption, access controls, and authentication mechanisms to ensure security**. The service uses a variety of protocols such as SFTP, FTP and AS2, thus supporting **TR-8: Provide APIs and connectors to integrate with multiple tools and services.** It implements isolation mechanisms with help of logical directories and IAM roles, thus supporting multi-tenancy for **TR-10: Support multi-tenancy by implementing isolation mechanisms for concurrent service instances.**
- AWS Lambda - AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. Lambda will automatically manage the compute resources for you, so you don't have to worry about provisioning or managing servers. The application uses lambda extensively - first lambda function is used to provide a custom authentication mechanism with AWS Secrets Manager thus helping with **TR-6: Implement encryption, access controls, and authentication mechanisms to ensure security** and **TR-11: Implement user authentication and validation mechanisms for secure access.** The second lambda function helps update the virus definitions on a regular basis via updates from the internet and inference drawn from machine learning model, thus fulfilling **TR-14: Integrate machine learning algorithms for improved fraud detection accuracy.** The third lambda function scans the files in the AWS S3 bucket for malwares and classifies them as such. It also sends data to AWS Sagemaker for inferences and better outcomes. Lambda has automatic resource allocation in order provide greater speed and higher performance thus helping complete **TR-2: Design the architecture to optimize resource allocation for high performance.**

  Trade Off - We decide to use a custom authentication function as opposed to a normal authentication, as a application with critical data needs to have strong authentication mechanisms. While this creates a cost and performance tradeoff, security is the most critical aspect of this application and as such we decide to prioritize that for **tradeoff 3**
- AWS Secrets Manager - AWS Secrets Manager is a cloud service that helps you manage, store, and retrieve secrets throughout their lifecycles. Secrets are sensitive

pieces of information, such as passwords, API keys, and certificates, that are used to access resources and applications. This service works with AWS Lambda to provide a custom authentication mechanism in the application, storing user credentials and access tokens for each tenant. This service helps fulfill **TR-11: Implement user authentication and validation mechanisms for secure access** and **TR-10: Support multi-tenancy by implementing isolation mechanisms for concurrent service instances.** Due to built in security features by Amazon and use of IAM roles, it also enables **TR-16: Define and enforce data and privacy policies to protect user data**

- AWS S3 Bucket - An Amazon S3 bucket is a storage repository that stores data objects at any scale. You can store any type of data in an S3 bucket, including text, images, videos, and audio. S3 buckets are highly durable and secure, and they can be accessed from anywhere in the world. AWS provide in built mechanism for resource allocation due to size and diversity of files being stored in S3 in order to enable high transfer speeds. This helps accomplish **TR-2: Design the architecture to optimize resource allocation for high performance.** It implements mechanism to use redundancy using data replication, erasure coding and storage classes, thus giving high availability of client files in the application and completing **TR-3: Use redundancy, load balancing, and failover mechanisms to ensure high availability.** It supports wide variety of OS like Windows Servers and RHEL 6+, and also provides variety of SDKs and APIs to ensure **TR-9: Ensure compatibility with various platforms and operating systems for multi-platform and operating systems support**
- AWS Quicksight - Amazon QuickSight is a cloud-based business intelligence (BI) service that allows you to create and deliver data insights to everyone in your organization. It provides a unified platform for data analysis and visualization. The application uses Amazon quicksight to gather inferences from the data generated by lambda function scanning the files. This can be used for analysis and infer statistics for training as per **TR-18: Use data analytics to analyze and infer statistics from logged data for system improvement and training**
- AWS Sagemaker - Amazon SageMaker is a fully managed service that helps developers and data scientists build, train, and deploy machine learning (ML) models quickly and easily. The application uses Sagemaker in order to train the malware detection system to reduce false positives and negatives based on results from past data. This improved model is again deployed using Lambda for better results. Sagemaker coupled with Lambda function **TR-14: Integrate machine learning algorithms for improved fraud detection accuracy.**
- AWS Disaster Recovery - AWS DR services can be used to recover data and applications from a variety of sources, including on-premises infrastructure, cloud-based infrastructure, and other AWS services. The AWS DR service is connected to S3 bucket storing the client files as well as Lambda function used for scanning them in order to ensure both the files and the model are highly available. It is used to **TR-13: Design and test disaster recovery and backup mechanisms to ensure data availability.**
  Trade Off - We decided to include a service for managing backups and failover mechanisms despite the cost over head as the application stored critical data and it had

to be ensured that the data loss in case of an incident should be minimized, thus addressing **trade off 2**

- Amazon Eventbridge schedule - Amazon EventBridge Scheduler is a serverless scheduler that allows you to schedule tasks at any time without having to provision or manage underlying infrastructure. It supports over 200 AWS services and more than 6,000 APIs. You can use EventBridge Scheduler to schedule one-time or recurrent tasks that trigger over 200 services with more than 6,000 APIs. It helps **TR-9: Ensure compatibility with various platforms and operating systems for multi-platform and operating systems support**
- AWS Codebuild: AWS CodeBuild is a fully managed continuous integration (CI) service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don't need to provision, manage, and scale your own build servers. CodeBuild scales on demand to meet your build needs. It **TR-1: Implement auto-scaling to accommodate time varying workloads.**
- Amazon ECR: Amazon Elastic Container Registry (ECR) is a fully managed container registry that lets you store, manage, and deploy Docker images and Open Container Initiative (OCI) images. It is a highly scalable and secure service that can be used to store and manage images for any number of applications. It **TR-1: Implement auto-scaling to accommodate time varying workloads and ha**s automatic container allocation as per need in order to maintain high performance thus fulfilling **TR-2: Design the architecture to optimize resource allocation for high performance.**
- AWS Cloudwatch - Amazon CloudWatch is a monitoring and observability service that provides a comprehensive view of your AWS resources and applications. It collects and monitors metrics, logs, and events from your AWS resources and applications, and provides you with the insights you need to troubleshoot issues, optimize performance, and improve your overall operational health. **TR-15: Implement logging and monitoring systems for analytics and performance tracking.**

5.5 Design Principles and Best Practices Used

- Best Practices Implemented
    - Operational Excellence
        - Evolve: The application is designed to continuously evolve. It runs detection algorithm on the files, gathers inferences from the same and uses it to improve the performance of the model. Using the AWS CodeBuild and AWS Sagemaker, the application will evolve based on the performance of the past model
    - Security
        - Identity and Access management - The custom authentication mechanism uses AWS Secrets Manager inorder to store user details such as passwords and auth tokens. The lambda function has checks for IAM roles for each user post authentication

        - Detection - The logs are captured at multiple instances for processing and detection. They are captured by AWS Kinesis for initial real time processing and subsequently by AWS Quicksight. The logs can be gathered from there for detection and prevention.
        - Data Protection - Plenty of data protection measures have been applied in the application. The files are stored in S3 buckets which are highly resilient and durable. They protect data at rest. AWS Transfer Family handles data in transit and is designed for secure file transfer.
        - Infrastructure Protection - The application uses AWS EC2 instances for Lambda function, as well as AWS ECR, both of which can be hardened and configured to provide additional security to the application
        - Incident Response - Detailed logging of every action taking place inside the application takes place at multiple points in time. Subsequently AWS Disaster Recovery is responsible for appropriate backup and recovery mechanisms.
    - Reliability
        - Workload architecture - The application is designed to be high scalable and handle time varying workloads. AUto scaling is implemented at various points in order to avoid bottlenecks in the system. Multiple backups are maintained in event of a failure to get the system back up as soon as possible
        - Failure Management - The application handles failure management upto a certain extent. It uses a managed service in order design and test backup and failure mechanism. It maintains multiple copies of the backup and disaster recovery mechanism
    - Performance
        - Compute and hardware - The application uses AWS ECR for configuration and management of containers. This service handles resources allocation, scaling the use of containers as per demand to maintain high performance
        - Data management - The application uses AWS S3 Storage. . Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Amazon S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of applications for companies all around the world.
    - Cost Optimization
        - Expenditure and usage awareness - The application uses combination of AWS Quicksight and cost allocation feature of AWS S3 Bucket to monitor expenditure and usage of the application.
- Design Principles Implemented
    - Operation Excellence

        - Use managed service - The application has chosen AWS as the preferred cloud provider and implements an AWS managed service wherever plausible.
    - Security
        - Implement a strong identity foundation - The application has a custom authentication mechanism in order to store and verify user credentials using AWS Secrets Manager
        - Protect Data in Transit and at Rest - The data is stored in S3 buckets and AWS Transfer family is used to securely transfer data from one service to the other.
        - Prepare for security events - The incident response and planning is handled by AWS Disaster Recovery for appropriate failover mechanisms and backup and recovery plans
    - Reliability
        - Automatically recover from failure - Detailed logging for key performance metrics take place, as well as backups for recovery in event of failure
        - Test recovery procedures - AWS DR is used to design and test recovery procedures in event that the application fails
        - Scale horizontally to increase aggregate workload availability - Auto scaling is implemented at multiple points in the application in order to ensure availability. AWS Lambda, Transfer Family and CodeBuild come with built in auto scaling functionality to ensure high availability
    - Performance
        - Use serverless architecture - The application has minimal use of servers by deploying AWS Lambda functions wherever possible.
    - Sustainability
        - Use managed services - Using managed services that can help minimize impact, such as automatically moving infrequently accessed data to cold storage with Amazon S3 Lifecycle configurations. The application uses AWS S3, AWS Codebuild and AWS Transfer to minimize unnecessary overhead for data at rest and in transit.

5.6 Tradeoffs revisited

Section 2.4, Ref[1], Ref[20]

1. Tradeoff 1 - Auto Scaling vs Cost Optimization

	Alternatives	
Objectives	Auto Scaling	No Scaling
High Availability	Yes	No

~~Cost Optimization~~	~~No~~	~~Yes~~

Between auto scaling and cost optimization, we have decide to prioritize Auto Scaling over Cost Optimization for following reasons

- Increased Workload and Traffic Variability: Malware detection applications often face unpredictable spikes in workload and traffic, especially during malware outbreaks or targeted attacks. Scalability allows the system to handle increased demand effectively, ensuring that the application remains responsive and capable of processing a growing number of requests without degradation in performance. This is crucial for maintaining the effectiveness of malware detection in real-time.
- Massive Data Ingestion: Malware detection applications process vast amounts of data, including network traffic, file uploads, and endpoint logs. Scalability ensures that the application can efficiently ingest and analyze this data without performance bottlenecks or delays, ensuring timely threat detection.
- Quick Response to Incidents: In the event of a malware outbreak or a sudden surge in malicious activities, a scalable architecture allows the malware detection system to rapidly allocate additional resources to analyze and mitigate the threats. This agility is crucial for minimizing the impact of malware incidents and responding swiftly to protect users and data.
- Rapidly Evolving Malware Landscape: Malware threats are constantly evolving, with new strains and attack techniques emerging regularly. A scalable malware detection application can effectively handle this dynamic environment by seamlessly expanding its processing capacity to accommodate new data volumes and analysis tasks.
- Future Growth and Flexibility: Prioritizing scalability positions the malware detection application for future growth. As the user base expands or new features are added, the system can easily adapt by scaling horizontally or vertically. This flexibility is essential for accommodating changing business requirements and technological advancements without undergoing significant architectural overhauls.

2. Tradeoff 2: Use redundancy, load balancing, and failover mechanisms to ensure high availability versus cost optimization

	Alternatives	
Objectives	Redundancy	Cost Optimization
High Availability	Yes	No
~~Cost Optimization~~	~~No~~	~~Yes~~

We have chosen to implement load balancing, redundancy and incident response/failoer mechanism over cost optimization for following reasons -

- Continuous Service Availability: Malware detection is a mission-critical function, and downtime can have severe consequences, including increased vulnerability to cyber threats. Redundancy, load balancing, and failover mechanisms work together to minimize downtime by ensuring that if one component or server fails, another can seamlessly take over. This continuous availability is crucial for providing uninterrupted protection against malware.
- Continuous Threat Protection: Malware attacks can strike at any time, and organizations need continuous protection to mitigate these threats. High availability ensures that the malware detection application remains operational 24/7, providing uninterrupted protection against evolving threats.
- Competitive Advantage: In a competitive landscape, organizations rely on their ability to provide reliable and secure services. High availability differentiates organizations from competitors, demonstrating a commitment to uptime and data protection.
- Geographical Redundancy: Distributing the infrastructure across multiple geographical locations enhances redundancy and provides geographic diversity. In the event of a localized failure, such as a data center outage or a region-specific issue, the malware detection application can still operate from other locations. This geographic redundancy improves resilience and contributes to a higher level of availability.
- Fault Tolerance: Failover mechanisms play a crucial role in fault tolerance. In the event of a server or component failure, failover mechanisms automatically redirect traffic to healthy, redundant instances. This ensures that the application remains operational even when individual components experience issues. The ability to quickly and automatically switch to backup systems minimizes the impact of failures on the overall system.
- Mitigation of Single Points of Failure: Redundancy and failover mechanisms help eliminate single points of failure within the infrastructure. By having duplicate components and automatic failover processes in place, the system becomes less vulnerable to disruptions caused by hardware failures, software glitches, or other issues affecting individual components.

3. Tradeoff 3: Implement encryption, access controls, and authentication mechanisms to ensure security vs. Design the architecture to optimize resource allocation for high performance

	Alternatives	
Objectives	Encryption	Resource Allocation
High Security	Yes	No
~~High Performance~~	~~No~~	~~Yes~~

We have chosen to prioritize encryption, access control and authentication mechanisms for security as opposed to resource allocation for better performance for following reasons:

- Sensitivity of Data: Malware detection applications handle sensitive data, including network traffic, file uploads, and endpoint logs. This data may contain confidential information, such as intellectual property, financial records, and personal data. Implementing robust security measures, such as encryption, access controls, and authentication, is crucial to protect this sensitive data from unauthorized access, theft, or manipulation.
- Regulatory Compliance: Many industries have strict regulations regarding data security, such as HIPAA in healthcare and PCI DSS in finance. Implementing strong security measures helps organizations comply with these regulations and avoid potential penalties or legal repercussions.
- Risk Reduction in Multi-Tenant Environments: In a cloud environment where multiple users or organizations share resources, implementing security measures becomes even more critical. Encryption ensures that data is protected even in a shared infrastructure. Access controls and authentication mechanisms help maintain isolation between different users or tenants, reducing the risk of one entity inadvertently accessing or affecting the data and processes of another.
- Resilience Against Insider Threats: Robust access controls and authentication mechanisms contribute to resilience against insider threats—those arising from individuals within the organization who may have malicious intent. By limiting access based on roles and responsibilities, organizations can reduce the risk of insider misuse or abuse of sensitive data and functionalities.
- Long-Term Viability and Reputation: Security breaches can have severe consequences for the long-term viability and reputation of a malware detection application. Users and clients expect security as a fundamental aspect of the service. Investing in encryption, access controls, and authentication mechanisms is a proactive measure to build and maintain trust in the application's security posture.

6. Kubernetes Experimentation:

6.1 Experiment Design:

We will validate some aspects of our design using Kubernetes and observe how it satisfies some of our TRs of our architecture design.

The technical requirements covered in this section are:

TR-1: Implement auto-scaling to accommodate time varying workloads

- Auto-scaling ensures that the resources are allocated and unallocated as per the demand automatically. This reduces the need for manual intervention, simplifying management tasks of the team.
- In the experiment, HPA (Horizontal Pod Autoscaler) in Kubernetes reacts to changes in the workload in near real-time, providing dynamic and automatic adjustments to the number of pod instances. This adaptability is especially valuable for applications with

variable workloads. With more demands the number of pod instances are increased and during less demand the pod instances are decreased.

TR-2: Design the architecture to optimize resource allocation for high performance

- HPA enables dynamic scaling of the number of pod replicas in response to changes in demand. When the observed metrics (such as CPU utilization or custom metrics) indicate that additional resources are needed to maintain performance, HPA automatically scales up the number of pod replicas
- For example, we can set a target CPU utilization of 70% and specify a desired range between 50% and 80%. HPA uses these targets and thresholds to make scaling decisions, optimizing resource allocation to keep performance within acceptable bounds.
- In this experiment, we are setting the target CPU utilization to 50%.

TR-15: Implement logging and monitoring systems for analytics and performance tracking

- HPA continuously monitors the specified metrics, such as CPU utilization or custom metrics, for the pods in the target deployment or replica set. The metrics are collected from the underlying container runtime using a metrics server.
- In a broader monitoring system used for performance tracking and analytics, we can use the information provided by HPA and metrics-server performance and utilization metrics
- In this experiment, we simulate a monitoring system using ‘kubectl get hpa –watch’ command.

Experiment Brief:

- In the experiment, we will deploy a php-apache server which will be treated as our application server and an HPA instance for it using Kubernetes (on a local environment using minikube).
- We will generate load on this deployment and validate by observing how some of the technical requirements of our design are met as discussed.
- The minimum number of replicas that will be deployed is 1 and maximum number of replicas will be 10.
- The target CPU utilization that we are aiming to achieve is 50% and the autoscaling will occur using the HPA based on the CPU utilization.

Experiment Setup:

- We have installed minikube on our local windows operating system. The driver that the minikube cluster is using is docker. In Figure 6.1.1.1 we verify that minikube is up and running on our system in a docker container.

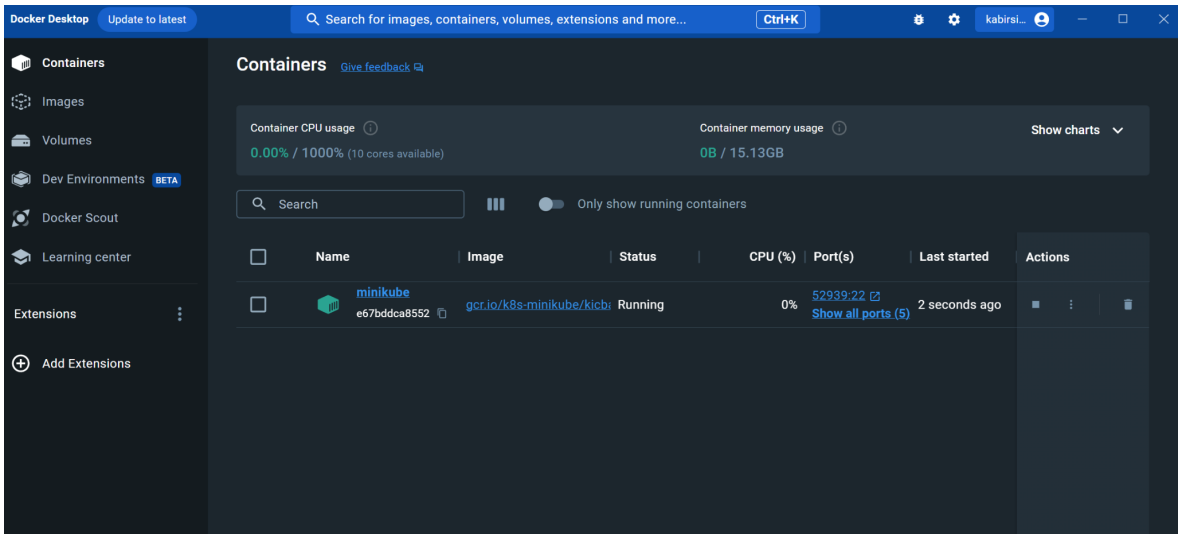


Figure 6.1.1.1

- kubectl is a command-line tool that is used to interact with Kubernetes. On our system, we have installed kubectl to interact with minikube. In Figure 6.1.1.2, we verify the kubectl installation and version.

Checking kubectl version

```
PS D:\Cloud Section 6\project> kubectl version --client --output=yaml
clientVersion:
  buildDate: "2023-09-13T09:35:49Z"
  compiler: gc
  gitCommit: 89a4ea3e1e4ddd7f7572286090359983e0387b2f
  gitTreeState: clean
  gitVersion: v1.28.2
  goVersion: go1.20.8
  major: "1"
  minor: "28"
  platform: windows/amd64
kustomizeVersion: v5.0.4-0.20230601165947-6ce0bf390ce3
```

Figure 6.1.1.2

- Now we verify whether our minikube node is healthy. For this we use ‘kubectl get nodes’ to check the active nodes and in this case, our minikube STATUS is ‘Ready’ indicating that it is in fact ready to accept any workloads.

Viewing the active nodes

```
PS D:\Cloud Section 6\project> kubectl get nodes
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   6d19h   v1.28.3
PS D:\Cloud Section 6\project>
```

Figure 6.1.1.3

- We also verify whether there are any pods or deployments running in our minikube node before we begin so that we can start with a clean slate.

Viewing the pods and deployments

```
PS D:\Cloud Section 6\project> kubectl get pods
No resources found in default namespace.
PS D:\Cloud Section 6\project> kubectl get deploy
No resources found in default namespace.
```

Figure 6.1.1.4

Experiment Walkthrough:

- First we create a manifest file for our application server as shown below. In the file we describe the configurations for our application deployment as well as the corresponding service for that deployment.
- Here the image we are using for this application is ‘registry.k8s.io/hpa-example’ which is a php-apache server that returns ‘OK!’ when sending a GET request to / route.
- The maximum CPU limit for the container to 450 milliCPU (m) and the amount of CPU resources that the container requests, set to 100 milliCPU (m).
- The application is open to port 80 within the Kubernetes cluster.
- The name of both the deployment and the service is ‘malware-threat-detection’.

malware-threat-detection.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: malware-threat-detection
spec:
  selector:
    matchLabels:
      run: malware-threat-detection
  template:
    metadata:
      labels:
        run: malware-threat-detection
    spec:
      containers:
      - name: malware-threat-detection
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 450m
          requests:
            cpu: 100m
---
apiVersion: v1
kind: Service
metadata:
  name: malware-threat-detection
  labels:
    run: malware-threat-detection
spec:
  ports:
  - port: 80
  selector:
    run: malware-threat-detection
```

Ref[19]

- We use this manifest file to create the deployment and the service using the command ‘kubectl apply -f .\malware-threat-detection.yaml’ as shown in Figure 6.1.1.5.

Creating deployment using malware-threat-detection.yaml

```
PS D:\Cloud Section 6\project> kubectl apply -f .\malware-threat-detection.yaml
deployment.apps/malware-threat-detection created
service/malware-threat-detection created
```

Figure 6.1.1.5

- We verify the creation of deployment and the corresponding service and pod as shown in Figure 6.1.1.6 (a), (b) and (c).

Deployment created

```
PS D:\Cloud Section 6\project> kubectl get deploy
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
malware-threat-detection   1/1     1            1           71s
```

Figure 6.1.1.6 (a)

Service created:

```
PS D:\Cloud Section 6\project> kubectl get service
NAME                       TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes                 ClusterIP   10.96.0.1      <none>        443/TCP   15m
malware-threat-detection   ClusterIP   10.105.58.83   <none>        80/TCP    15m
```

Figure 6.1.1.6 (b)

Pods in the deployment

```
PS D:\Cloud Section 6\project> kubectl get pods
NAME                                       READY   STATUS    RESTARTS   AGE
malware-threat-detection-6cbbf5788-dvclb   1/1     Running   0          5m14s
```

Figure 6.1.1.6 (c)

- Now we enable metrics-server addon in the minikube. This addon collects resource usage metrics from the nodes and pods which will be later used by HPA for making scaling decisions.
- We enable it and verify whether it is enabled as shown in Figure 6.1.1.7 (a) and Figure 6.1.1.7 (b)

Enabled metrics-server addon in minikube for hpa:

```
PS D:\Cloud Section 6\project> minikube addons enable metrics-server
W1123 20:12:15.812299   12340 main.go:291] Unable to resolve the current Docke
eta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json
💡  metrics-server is an addon maintained by Kubernetes. For any concerns cont
You can view the list of minikube maintainers at: https://github.com/kubernete
    ▪ Using image registry.k8s.io/metrics-server/metrics-server:v0.6.4
🌟  The 'metrics-server' addon is enabled
```

Figure 6.1.1.7 (a)

Verifying the addon being enable

```
PS D:\Cloud Section 6\project> minikube addons list
W1123 20:15:17.231254   48492 main.go:291] Unable to resolve the current Docker CLI context "
eta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system ca
|------------------------------|----------|--------------|--------------------------------|
|         ADDON NAME           | PROFILE  |    STATUS    |           MAINTAINER           |
|------------------------------|----------|--------------|--------------------------------|
| ambassador                   | minikube | disabled     | 3rd party (Ambassador)         |
| auto-pause                   | minikube | disabled     | minikube                       |
| cloud-spanner                | minikube | disabled     | Google                         |
| csi-hostpath-driver          | minikube | disabled     | Kubernetes                     |
| dashboard                    | minikube | disabled     | Kubernetes                     |
| default-storageclass         | minikube | enabled ✅   | Kubernetes                     |
| efk                          | minikube | disabled     | 3rd party (Elastic)            |
| freshpod                     | minikube | disabled     | Google                         |
| gcp-auth                     | minikube | disabled     | Google                         |
| gvisor                       | minikube | disabled     | minikube                       |
| headlamp                     | minikube | disabled     | 3rd party (kinvolk.io)         |
| helm-tiller                  | minikube | disabled     | 3rd party (Helm)               |
| inaccel                      | minikube | disabled     | 3rd party (InAccel             |
|                              |          |              | [info@inaccel.com])            |
| ingress                      | minikube | disabled     | Kubernetes                     |
| ingress-dns                  | minikube | disabled     | minikube                       |
| inspektor-gadget             | minikube | disabled     | 3rd party                      |
|                              |          |              | (inspektor-gadget.io)          |
| istio                        | minikube | disabled     | 3rd party (Istio)              |
| istio-provisioner            | minikube | disabled     | 3rd party (Istio)              |
| kong                         | minikube | disabled     | 3rd party (Kong HQ)            |
| kubeflow                     | minikube | disabled     | 3rd party                      |
| kubevirt                     | minikube | disabled     | 3rd party (KubeVirt)           |
| logviewer                    | minikube | disabled     | 3rd party (unknown)            |
| metallb                      | minikube | disabled     | 3rd party (MetalLB)            |
| metrics-server               | minikube | enabled ✅   | Kubernetes                     |
| nvidia-device-plugin         | minikube | disabled     | 3rd party (NVIDIA)             |
| nvidia-driver-installer      | minikube | disabled     | 3rd party (Nvidia)             |
| nvidia-gpu-device-plugin     | minikube | disabled     | 3rd party (Nvidia)             |
| olm                          | minikube | disabled     | 3rd party (Operator Framework) |
| pod-security-policy          | minikube | disabled     | 3rd party (unknown)            |
| portainer                    | minikube | disabled     | 3rd party (Portainer.io)       |
| registry                     | minikube | disabled     | minikube                       |
| registry-aliases             | minikube | disabled     | 3rd party (unknown)            |
| registry-creds               | minikube | disabled     | 3rd party (UPMC Enterprises)   |
| storage-provisioner          | minikube | enabled ✅   | minikube                       |
| storage-provisioner-gluster  | minikube | disabled     | 3rd party (Gluster)            |
| storage-provisioner-rancher  | minikube | disabled     | 3rd party (Rancher)            |
| volumesnapshots              | minikube | disabled     | Kubernetes                     |
|------------------------------|----------|--------------|--------------------------------|
```

Figure 6.1.1.7 (b)

- Now we create a manifest file to create a Horizontal Pod Autoscaler for our application as shown in ‘hpa.yaml’.
- The minimum number of replicas (minReplicas) that the application can scale down to is set to 1 and the maximum number (maxReplicas) is set to 10.
- The resource that the HPA will use for scaling is CPU and the average CPU Utilization is set to 50% as seen in the manifest file (averageUtilization).

hpa.yaml

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: malware-threat-detection
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: malware-threat-detection
  minReplicas: 1
```

```
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

- We provision the HPA using the manifest file as shown in Figure 6.1.1.8 using ‘kubectl apply -f .\hpa.yaml’.

Provisioned HPA for the deployment:

```
PS D:\Cloud Section 6\project> kubectl apply -f .\hpa.yaml
horizontalpodautoscaler.autoscaling/malware-threat-detection created
```

Figure 6.1.1.8

- We verify that the HPA is provisioned using the ‘kubectl get hpa’ command as shown in Figure 6.1.1.9. It also shows the current utilization metric and target utilization metric of the CPU (1%/50%). Since there is no load on the application, we can see that the current number of replicas of the application are minimum (1) and the current CPU Utilization is 1%.

Viewing HPA (without load):

```
PS D:\Cloud Section 6\project> kubectl get hpa
NAME                       REFERENCE                             TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
malware-threat-detection   Deployment/malware-threat-detection   1%/50%    1         10        1          2m9s
```

Figure 6.1.1.9

- We simulate monitoring over the HPA using ‘kubectl get hpa --watch’ command. The --watch flag will continuously monitor any changes to the HPA in real time.
- Using this we can monitor how HPA reacts to changes in resource metrics and adjusts the number of replicas.

Monitoring HPA

```
PS C:\Users\kabir> kubectl get hpa --watch
NAME                       REFERENCE                                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
malware-threat-detection   Deployment/malware-threat-detection   1%/50%    1         10        1          29m
```

Figure 6.1.1.10

- Now since the application deployment, service and the HPA is provisioned, we can proceed with the load generation.
- We create a load generator as shown in Figure 6.1.1.11 in another windows terminal instance. We create a pod (which will get deleted upon termination) using a busybox image that generates HTTP traffic to the service of our application (malware-threat-detection). It uses a while loop to continuously send GET requests to our service thereby increasing load on the application. We can also observe the output of each request (‘OK!’).

Generating load:

```
PS C:\Users\kabir> kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do
 wget -q -O- http://malware-threat-detection; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Figure 6.1.1.11

- Upon monitoring the HPA, we can observe that the current CPU utilization spikes up to 298%.
- As time passes, the HPA scales up to accommodate the load in order to reach the target CPU Utilization by increasing the number of replicas. Due to this, the current CPU utilization decreases as more and more pods are provisioned, stabilizing at 34% with the maximum number of pods provisioned (10). This can be observed in Figure 6.1.1.12.

Monitoring HPA during load generation:

```
PS C:\Users\kabir> kubectl get hpa --watch
NAME                       REFERENCE                                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
malware-threat-detection   Deployment/malware-threat-detection   1%/50%    1         10        1          29m
malware-threat-detection   Deployment/malware-threat-detection   298%/50%   1         10        1          31m
malware-threat-detection   Deployment/malware-threat-detection   298%/50%   1         10        4          31m
malware-threat-detection   Deployment/malware-threat-detection   298%/50%   1         10        6          31m
malware-threat-detection   Deployment/malware-threat-detection   165%/50%   1         10        6          32m
malware-threat-detection   Deployment/malware-threat-detection   103%/50%   1         10        6          33m
malware-threat-detection   Deployment/malware-threat-detection   103%/50%   1         10        10         33m
malware-threat-detection   Deployment/malware-threat-detection   71%/50%    1         10        10         34m
malware-threat-detection   Deployment/malware-threat-detection   62%/50%    1         10        10         35m
malware-threat-detection   Deployment/malware-threat-detection   60%/50%    1         10        10         36m
malware-threat-detection   Deployment/malware-threat-detection   63%/50%    1         10        10         37m
malware-threat-detection   Deployment/malware-threat-detection   34%/50%    1         10        10         38m
```

Figure 6.1.1.12

- We also verify the condition of pods during load generation. In Figure 6.1.1.6 (c), we observed that only one pod was provisioned at provisioning of the deployment.
- Now in Figure 6.1.1.13 we can observe that 10 pods were provisioned by the HPA to accommodate the load for our application. Additionally we can also observe that the load generator pod is also up and running.

Number of pods after load generation

```
PS C:\Users\kabir> kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
load-generator                              1/1     Running   0          6m40s
malware-threat-detection-6cbbf5788-9fkcq    1/1     Running   0          5m37s
malware-threat-detection-6cbbf5788-cdjbw    1/1     Running   0          5m52s
malware-threat-detection-6cbbf5788-dvclb    1/1     Running   0          48m
malware-threat-detection-6cbbf5788-fgjwh    1/1     Running   0          3m52s
malware-threat-detection-6cbbf5788-fjvb7    1/1     Running   0          5m37s
malware-threat-detection-6cbbf5788-gswgh    1/1     Running   0          5m52s
malware-threat-detection-6cbbf5788-kx2lw    1/1     Running   0          3m52s
malware-threat-detection-6cbbf5788-lc4h4    1/1     Running   0          3m52s
malware-threat-detection-6cbbf5788-lp98b    1/1     Running   0          3m52s
malware-threat-detection-6cbbf5788-rdq4l    1/1     Running   0          5m52s
```

Figure 6.1.1.13

- At this point let us stop the load generation by terminating the load generator. Upon monitoring, we can observe that there is a drop in the CPU utilization as there is no load.
- With time, HPA scales down the application from 10 to 1 as shown in Figure 6.1.1.14.

Increase in number of replicas after load generation stabilizing at 34%. Stopping load generation at that point and scaling down occurs.

```
PS C:\Users\kabir> kubectl get hpa --watch
NAME                       REFERENCE                             TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
malware-threat-detection   Deployment/malware-threat-detection   1%/50%    1         10        1          29m
malware-threat-detection   Deployment/malware-threat-detection   298%/50%   1         10        1          31m
malware-threat-detection   Deployment/malware-threat-detection   298%/50%   1         10        4          31m
malware-threat-detection   Deployment/malware-threat-detection   298%/50%   1         10        6          31m
malware-threat-detection   Deployment/malware-threat-detection   165%/50%   1         10        6          32m
malware-threat-detection   Deployment/malware-threat-detection   103%/50%   1         10        6          33m
malware-threat-detection   Deployment/malware-threat-detection   103%/50%   1         10        10         33m
malware-threat-detection   Deployment/malware-threat-detection   71%/50%    1         10        10         34m
malware-threat-detection   Deployment/malware-threat-detection   62%/50%    1         10        10         35m
malware-threat-detection   Deployment/malware-threat-detection   60%/50%    1         10        10         36m
malware-threat-detection   Deployment/malware-threat-detection   63%/50%    1         10        10         37m
malware-threat-detection   Deployment/malware-threat-detection   34%/50%    1         10        10         38m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        10         39m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        10         43m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        7          43m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        7          44m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        1          44m
```

Figure 6.1.1.14

- We can also verify the number of pods provisioned at this point. As shown in Figure 6.1.1.15, we can see that the load generator is stopped and deleted and only one pod of our application is present as the others were deprovisioned by the HPA.

Number of pods after stopping load generation.

```
PS C:\Users\kabir> kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
malware-threat-detection-6cbbf5788-dvclb   1/1     Running   0          58m
PS C:\Users\kabir>
```

Figure 6.1.1.15

6.2 Workload Generation with Locust

- To simulate our application we are using a php-apache image that responds ‘OK!’ to GET requests.
- The load generator in section 6.1 is a pod that acts as a client and runs an infinite loop sending requests to the application.
- The load-generator pod is a way of quick and straightforward load testing to check how the application handles basic traffic. It is easy to set up and minimal configuration is required.
- We can make basic customization to this load generation configuration such as adjusting the sleep time interval or modifying the URL.
- Overall it is a very light-weight solution for resource constraint environments or simple scenarios.

- But for this experiment, we require comprehensive reporting and analysis of results.
- The main advantage that Locust offers for our experiment is that it generates detailed reports and charts, aiding in result interpretation and provides insights into response times, failures, and other metrics.

- For this setup, we will use the following python script:

locustfile.py

```python
import time
from locust import HttpUser, task, between

class QuickstartUser(HttpUser):
    wait_time = between(1, 5)

    @task
    def hello_world(self):
        self.client.get("/")
```

- This is a basic locust script to generate load on the application URL.
- The wait_time variable specifies the wait time between task executions for each simulated user. In this case, it's set to a random value between 1 and 5 seconds. This introduces variability to simulate a more realistic user scenario where users do not perform tasks at fixed intervals.
- When we run this script using ‘locust’ command, we will be able to access a web-interface hosted at http://localhost:8089 using which we will customize the number of most concurrent users to 23 with the spawn rate of 1 user per second as shown in Figure 6.1.1.18.

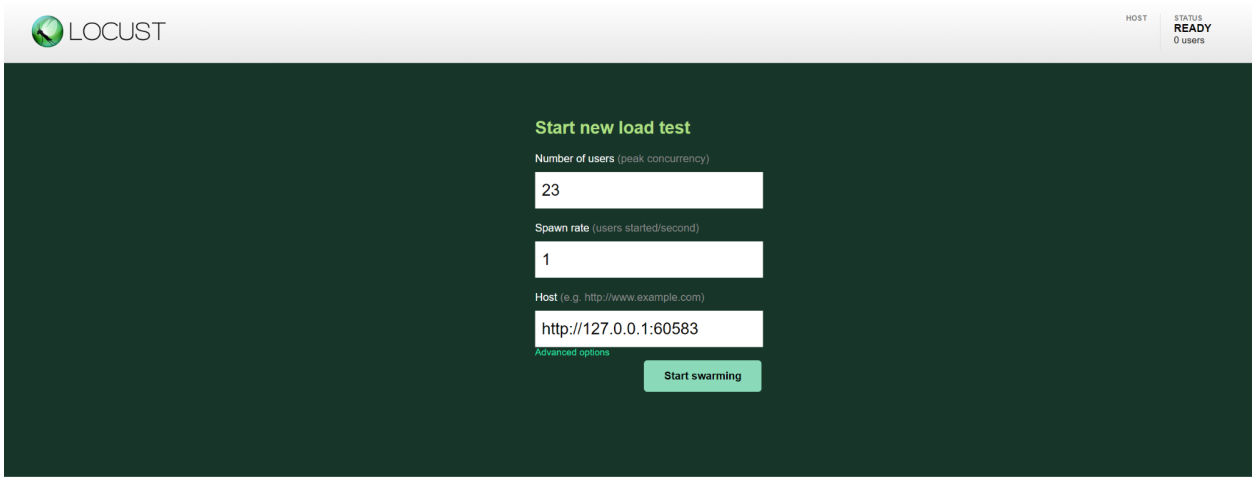


Figure 6.1.1.16

- The URL of the application can be obtained by using ‘minikube service <service-name>’ command as shown in Figure 6.1.1.17.

Getting application URL

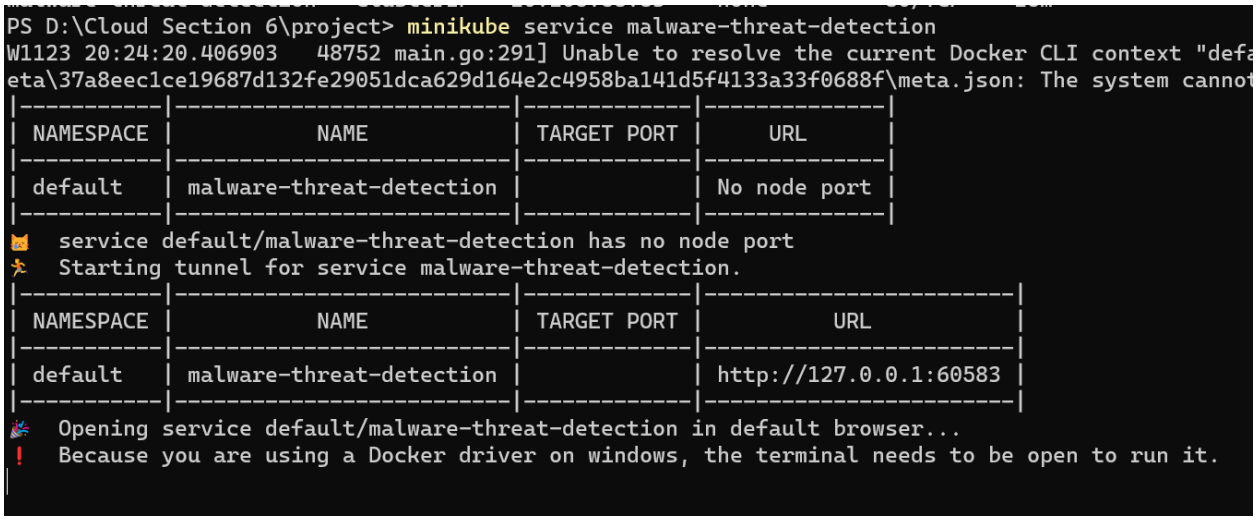


Figure 6.1.1.17

6.3 Analysis of the results

- We begin the load generation using Locust for our application.
- On checking the HPA during load generation, we observe that the HPA scales the application up from 1 replica to 10 replicas to stabilize the current CPU utilization 45% and accommodate the load as shown in Figure 6.1.1.18.

Autoscaling (up) during load generation:

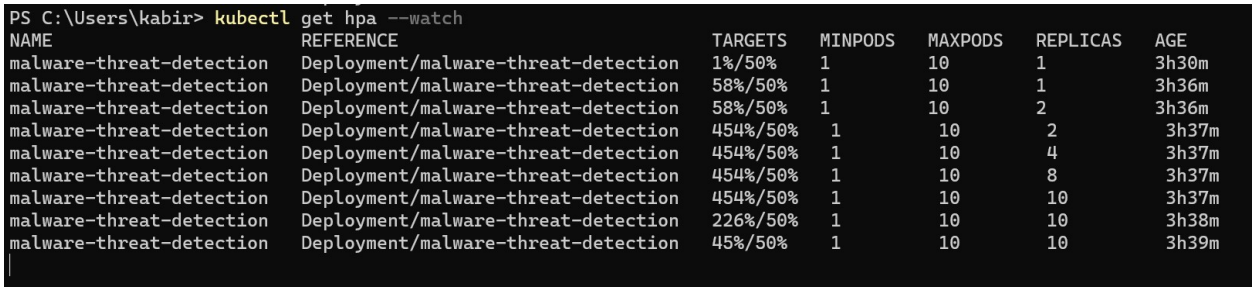


Figure 6.1.1.18

- After stopping the load generation we can observe that the HPA scales down and the number of replicas are reduced back to 1 replica since there is no load on the application as shown in Figure 6.1.1.19

Autoscaling (down) after stopping load generation

```
PS C:\Users\kabir> kubectl get hpa --watch
NAME                       REFERENCE                             TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        1          3h30m
malware-threat-detection   Deployment/malware-threat-detection   58%/50%    1         10        1          3h36m
malware-threat-detection   Deployment/malware-threat-detection   58%/50%    1         10        2          3h36m
malware-threat-detection   Deployment/malware-threat-detection   454%/50%   1         10        2          3h37m
malware-threat-detection   Deployment/malware-threat-detection   454%/50%   1         10        4          3h37m
malware-threat-detection   Deployment/malware-threat-detection   454%/50%   1         10        8          3h37m
malware-threat-detection   Deployment/malware-threat-detection   454%/50%   1         10        10         3h37m
malware-threat-detection   Deployment/malware-threat-detection   226%/50%   1         10        10         3h38m
malware-threat-detection   Deployment/malware-threat-detection   45%/50%    1         10        10         3h39m
malware-threat-detection   Deployment/malware-threat-detection   43%/50%    1         10        10         3h42m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        10         3h43m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        10         3h46m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        9          3h47m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        9          3h47m
malware-threat-detection   Deployment/malware-threat-detection   1%/50%     1         10        1          3h48m
```

Figure 6.1.1.19

- In Locust's interface, we can visually observe how the application is responding to the load in real time during the load generation as shown in Figure 6.1.1.21 (a) and Figure 6.1.1.21 (b)

Visualization using Locust

Figure 6.1.1.20 (a)

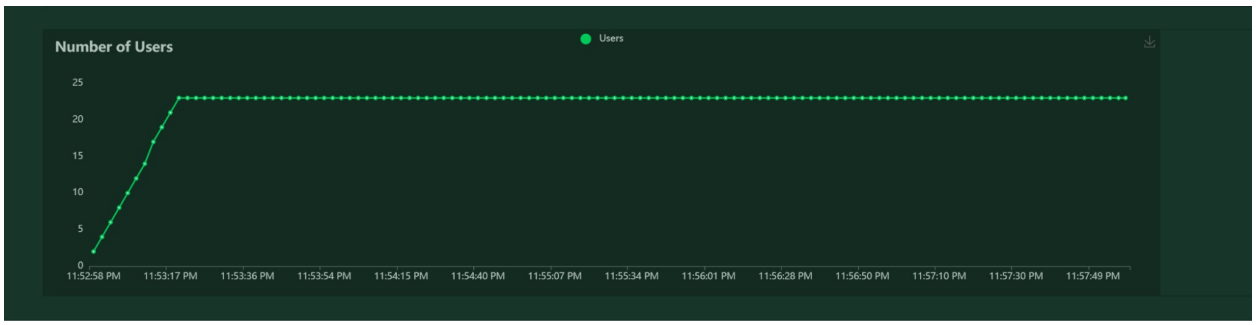


Figure 6.1.1.20 (b)

- Locust visualizes the distribution of response times in the form of percentiles (50th and 95th), allowing us to see how response times are distributed across different percentiles.
- Request Rate Chart displays the rate at which requests are being made per second.
- Response Time Chart shows the response times of requests over time.
- Number of Users Chart shows the number of active users over time.

The following TRs can be verified through these results:

TR-15: Implement logging and monitoring systems for analytics and performance tracking

- As discussed in section 6.1 HPA is continuously monitoring the specified metrics (i.e CPU utilization or custom metrics, for the pods in the target deployment or replica set). The metrics are collected from the underlying container runtime using a metrics server which was enabled in the minikube as an addon.
- In this experiment, we simulate a monitoring system using 'kubectl get hpa –watch' command in a less broad sense to monitor utilization metrics and scaling of replicas.

TR-1: Implement auto-scaling to accommodate time varying workloads

- The HPA is reacting to changes in the workload in near real-time, providing dynamic and automatic adjustments to the number of pod instances.
- In Locust we have set wait time range from 1-5 seconds for a user and the time is selected randomly for each user. The spawn rate increments one user per second [as observed in the Locust interface Figure 6.1.1.20 (b)] and in the end we stop the load generation. The application automatically scales up and down to accommodate such time varying workloads correctly as per Figure 6.1.1.18 and Figure 6.1.1.19.

TR-2: Design the architecture to optimize resource allocation for high performance

- From Figure 6.1.1.18 and Figure 6.1.1.19 we can observe that the increase in replicas occur in response to when the current CPU utilization exceeds the target utilization.
- Similarly when the load generation is stopped, the number of replicas decrease due to decrease/lack of load and replenish the resources that were used before while still hitting the target CPU utilization. Thus the resources are utilized as per the demands.
- Observing the Locust interface, we see that the maximum number of concurrent users is reached in the first 23 seconds as the spawn rate was set to 1 user per second.
- Since at that point in time only one replica was allocated, the response time was close to 7000-8000 ms for a total 2-3 requests per second (95th percentile). After some time, response times started stabilizing as the pods were being scaled up and after scaling, the response time dropped and stabilized around 3000 ms and also proceeded to decrease with time indicating that the application had scaled up. Also around that point we can also observe that the total number of requests per second has reached close to 6 requests per second. So for a high range of total number of requests with 23

concurrent active users (with random wait time in range 1-5 seconds), the application scaled and provided a low range of response time in the end.

- This indicates that the resources were allocated and high performance was achieved while at the same time, target CPU utilization was achieved as observed in Figure 6.1.1.18 and Figure 6.1.1.19.

10 Conclusion

10.1 Lessons Learned

Following lessons were learned during development of the project:

- **Cloud Infrastructure and Services**: We gained experience working with cloud infrastructure concepts like virtual machines, containers, and serverless computing. You'll also learn about cloud services for storage, networking, and data processing, such as Amazon S3, Amazon EC2, and Google Cloud Storage.
- **AWS Well Architected Framework:** Understand the key core concepts and practices recommended and implemented by AWS Well Architected Framework
- **Architecture Design**: We learnt how to design a scalable and resilient architecture for your malware detection application. Explore microservices architecture, containerization (e.g., Docker), and orchestration tools (e.g., Kubernetes) for deploying and managing your application components.
- **Security Best Practices**: Implementing encryption for data in transit and at rest to enhance security. Understand access controls and authentication mechanisms to prevent unauthorized access to your application and data. Explore identity and access management (IAM) features provided by cloud platforms.
- **Cybersecurity Principles**: We gain a deeper understanding of cybersecurity principles, including threat modeling, risk management, and incident response. We learnt how to design secure systems, identify potential threats, and respond effectively to security breaches.
- **Cloud-based Security Solutions:** We gained exposure to cloud-based security solutions, such as cloud firewalls, intrusion detection systems, and web application firewalls. We learnt how to integrate these solutions with your malware detection application to enhance overall security.

- **Cloud Security Best Practices**: We learnt about cloud security best practices, such as least privilege, identity and access management (IAM), and data encryption. We can apply these practices to secure your cloud-based malware detection application.
- **Collaboration and Teamwork**: Work collaboratively to understand the importance of effective communication, task delegation, and project management.
- **Documentation and Communication**: Document architecture, design decisions, and code to facilitate future collaboration and understanding.

References:

1. https://chat.openai.com/
2. *https://veritis.com/wp-content/uploads/2022/04/AWS-vs-Azure-vs-GCP-Cloud-Cost-Comparison.pdf*
3. https://www.techtarget.com/searchaws/tip/Know-your-AWS-downtime-limits#:~:text=Set%20expectations%20for%20AWS%20downtime,year%20of%20allowable%20outage%20time.
4. https://www.pluralsight.com/resources/blog/cloud/cloud-security-comparison-aws-vs-azure-vs-gcp
5. https://aws.amazon.com/aws-transfer-family/
6. https://aws.amazon.com/s3/
7. https://aws.amazon.com/lambda/
8. https://aws.amazon.com/secrets-manager/

9. https://aws.amazon.com/eventbridge/
10. https://aws.amazon.com/codebuild/
11. https://aws.amazon.com/ecr/
12. https://aws.amazon.com/kinesis/
13. https://aws.amazon.com/cloudtrail/
14. https://aws.amazon.com/sagemaker/
15. https://aws.amazon.com/cloudwatch/
16. https://aws.amazon.com/quicksight/
17. https://docs.aws.amazon.com/wellarchitected/latest/framework/security.html
18. https://docs.aws.amazon.com/wellarchitected/latest/framework/reliability.html
19. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/
20. https://hbr.org/1998/03/even-swaps-a-rational-method-for-making-trade-offs