Cloud Migration Solution Design Document

Section 1 - Project Description

1.1 Project

Cars Web application cloud POC

1.2 Description

This document provides a solution design to migrate the Cars web application to AWS cloud.

1.3 Revision History

Date	Comment	Author
2021-08-18	First draft	Morné Snyman
2021-08-19	Second draft	Morné Snyman
2021-08-20	Third draft	Morné Snyman

Contents

Section 1 - Project Description
1.1 Project
1.2 Description
1.3 Revision History
Section 2 - Overview
2.1 Purpose
2.2 Scope
2.3 Requirements
2.3.1 Phases
Section 3 - System Architecture
3.1 Solution design technology stacks
3.2 Solution design decisions
3.3 Application modifications
3.4 Detailed phase breakdown
3.4.1 Phase 1 - Migration
3.4.2 Phase 2 - Global scalability
3.4.3 Phase 3 - Security improvements
Section 4 - Extra Design Features / Outstanding Issues
Section 5 – References
Section 6 – Glossary

Section 2 - Overview

2.1 Purpose

The purpose of this project is to design a solution to migrate the Cars web application to the AWS cloud platform.

2.2 Scope

The solution design is divided into 3 phases of deployment. Each phase will be outlined in this document and explained in more detail.

2.3 Requirements

The project requirements are as follows.

- Deploy the Cars web application on a cloud platform of choice
- Plan application changes required for the migration
- The solution must be cost optimized and scalable
- Enhanced security must be implemented

2.3.1 Phases

Below is a list of the solution deployment phases.

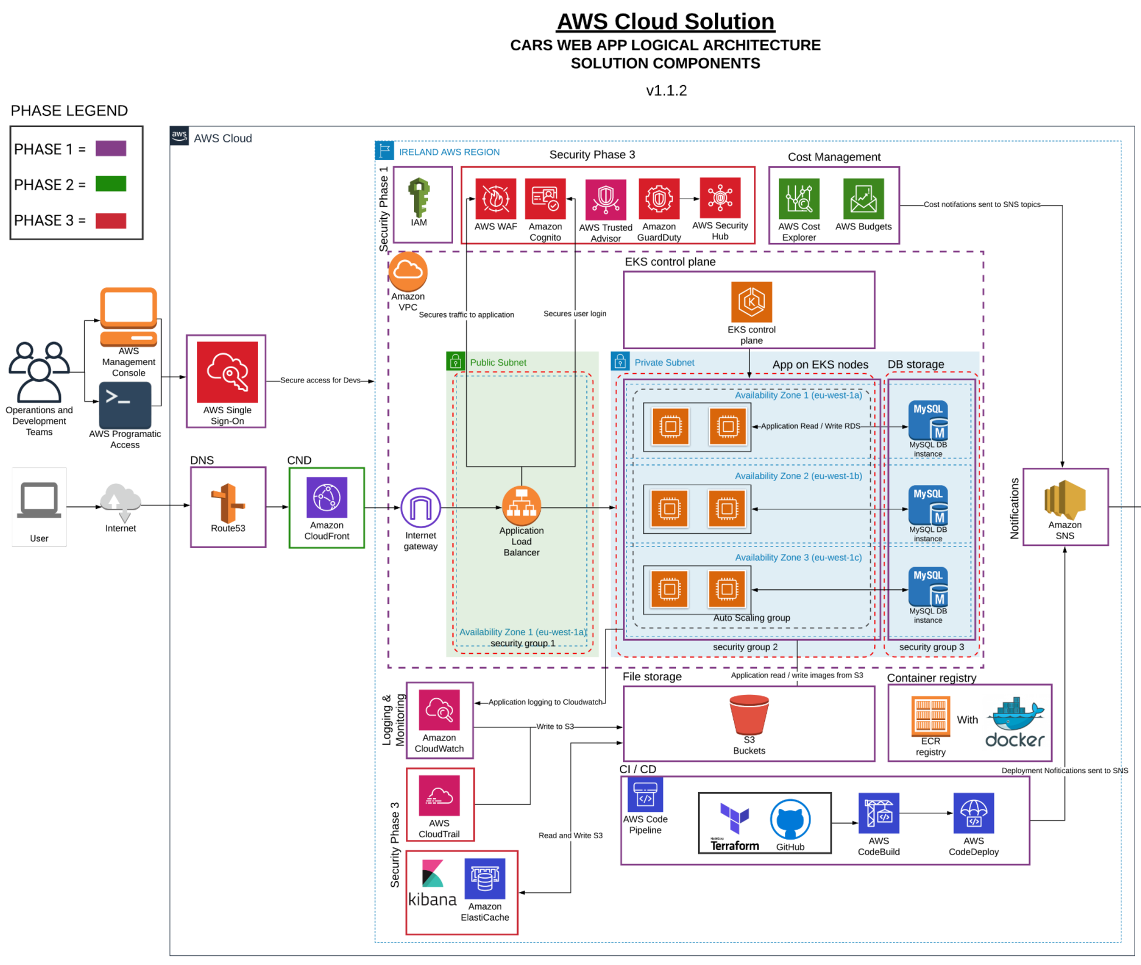
Phase	Description
1	Migrating the Cars web application to the AWS cloud - Minimal Viable Product (MPV)
2	Introduce global scalability
3	Security improvements

Section 3 - System Architecture

Below is a detailed design diagram of the solution design.

Each phase in the design is color coded to indicate which solutions will be implemented as part of each phase.

A separate image will be provided of the solution design diagram so that the diagram can be viewed in more detail.



3.1 Solution design technology stacks

- Amazon Web Services - Cars web application
    - AWS Single Sign-On
    - VPC
    - Internet Gateway
    - AWS Cognito
    - Route53
    - Cloudfront
    - EKS
    - ECS
    - S3
    - RDS
    - AWS Security Hub
    - Amazon GuardDuty
    - AWS WAF
    - AWS IAM
    - Kibana
    - Amazon Elasticache
    - AWS CloudTrial
    - Amazon Cloudwatch
    - AWS Cost explorer
    - AWS Budgets
    - SNS
    - AWS CodePipeline
    - AWS CodeBuild
    - AWS CodeDeploy

- GitHub - Version and source code control
- Terraform - Infrastructure as code
- Docker - Containers

3.2 Solution design decisions

The solution design decision was built around the AWS Well Architected framework which is used as a guideline on how to implement AWS best practices in an AWS environment.

Link: https://aws.amazon.com/architecture/well-architected

Cloud platform of choice

AWS was chosen as the cloud platform of choice mainly due to the large variety of services offered and wide support available.

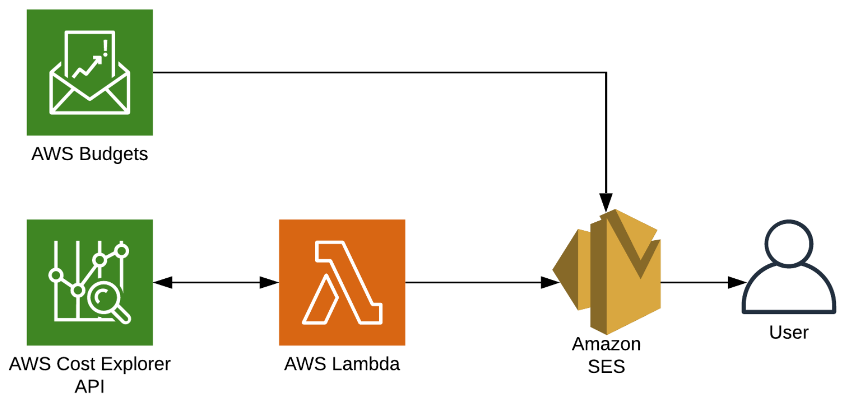
Cloud services

The decision was made to make use of as most of the AWS offerings available. This will also allow for easy integration between services as well as support and feature requests directly with AWS.

Cost Management

One of the primary concerns for this project is cost management. AWS Cost Explorer alongside AWS Budgets will be implemented to monitor costs. AWS SNS will be used to notify all parties for costing updates in the environment. AWS Cloudwatch Alerts will also be integrated to implement cost alerts if certain cost thresholds are reached.

The AWS Cost Calculator can also be used to determine the estimated cost for the initial deployment of the solution.
Link: https://calculator.aws/



AWS cost management and reporting solution

EKS

EKS was chosen for its high availability and scalability it offers by running the application in a containerised environment.

ECS

ECS will be used to store docker container images that will house the web application front end. The Cars web application will be containerized as docker images and deployed as containers on the EKS cluster nodes.

RDS

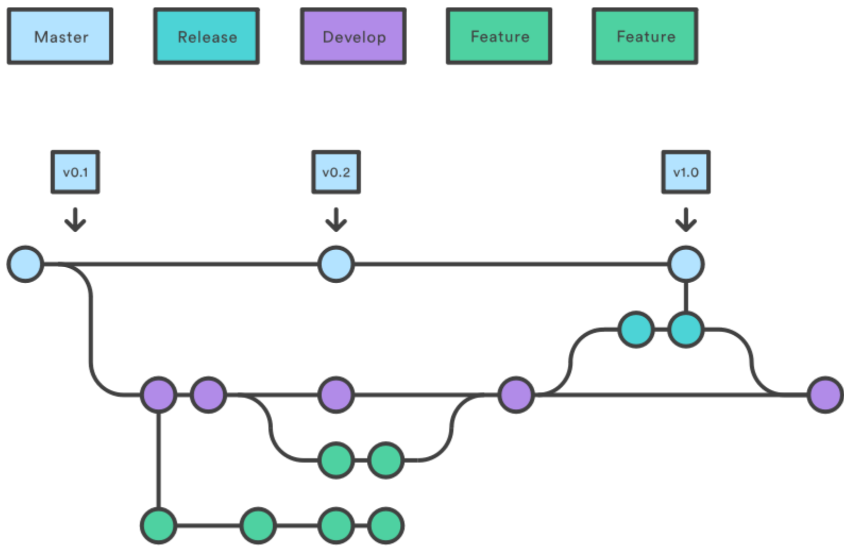
RDS was chosen for the hosting service for the database of the Cars web application. RDS allows for high availability and scalability, as well as integration into other AWS services. RDS offers a low level of management.

S3

S3 was chosen as the file storage. S3 offers the storage of unlimited amounts of data and is very cost effective. These features are great for storing the possibly large amounts of images for the Cars web application.

Docker
Container images will be built with docker. The Docker images will be built with AWS CodeBuild as part of the CI / CD CodePipeline and then deployed into AWS ECS.

Version and source code control
GitHub was selected as the service of choice for version and source code control of all code be it application or IaC. A private GitHub repository will be created and a branching structure will be implemented. Branches will consist of feature, dev, qa and production.



Git workflow with a branching structure of Feature, Develop, Release (QA) and Master.

Infrastructure as Code
Terraform was chosen as the primary tool for creating Infrastructure as Code. Terraform allows for more flexibility and future expansion in deploying to other cloud platforms. It is also currently seen as the tool of choice for IaC

CI / CD
The CI/CD implementation will be integrated from the initial deployment of the solution. This is to allow consistency of testing and deployments in the cloud environment. The CI / CD pipeline will make use of GitHub for code storage, CodeBuild to compile and test the application, CodeDeploy to deploy the application and related resources. All infrastructure resources will be built as IaC using Terraform.

Security
All of the security services and implementations will be AWS based. This is to allow for easy integration between other AWS services and also a wider range of support directly from AWS.

There will be a focus on security in regards to access to the environment by implementing AWS SSO and securing the Cars web application alongside the RDS database and S3 bucket for file storage.

AWS GuardDuty alongside AWS SecurityHub will be used as a compiled view of all security related issues and provide best practice blueprints.

AWS Cognito will be used to allow secure access to the application frontend. Last, AWS WAF will be used to secure all traffic to the application. Everything will be logged to Cloudwatch Logs and Cloudwatch Alerts will be implemented and send out notifications to all involved parties using AWS SNS.

Encryption
All objects stored in S3 will be encrypted in transfer and in rest with AWS KMS keys. Encryption will also be applied to all network traffic within the AWS network as well as all outside traffic coming into the AWS network.
The RDS databases will also be encrypted.

3.3 Application modifications

The application would need minor modifications for the migration. The application will be built into a Docker container image which would then be deployed within the EKS cluster nodes. Docker offers an easy service to containerize legacy applications at the same time.

Minor modifications such as configuring the database connection as well as configuring the file share as a s3 bucket will be required. The solution is designed as a lift and shift implementation with minor changes required to the application.

3.4 Detailed phase breakdown

3.4.1 Phase 1 - Migration

Below is a detailed description of each resource that will be deployed with the migration phase.

Region

The solution will be deployed within the Ireland region (eu-west-1) in AWS. This coincides with the most of the user traffic being from within Europe.

VPC

An Amazon VPC will be created inside the Ireland region alongside two subnets, 1 private and 1 public.

Subnets

- Public subnet

The public subnet will contain 1 availability zone with the Application Load Balancer. The Application Load Balancer will route traffic to resources in the private subnet.

- Private subnet

The private subnet will contain the Web application, Database and File storage. The private subnet contains 3 availability zones. The EKS nodes and Databases will be replicated across the 3 availability zones to allow for high availability and failover.

Security Groups

- Security group 1

This will be the Application Load Balancer security group. It will allow traffic from the internet and accept traffic from the EKS cluster where the Cars web application will be hosted on.

- Security group 2

This will be the EKS cluster security group. It will allow traffic from the Application Load Balancer and the RDS Database.

- Security group 3

This will be the security group for the RDS database. It will allow traffic from the EKS cluster.

AWS IAM

AWS Identity and Access Management (IAM) enables you to manage access to AWS services and resources securely. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.

AWS Single Sign-On

AWS Single Sign-On (AWS SSO) is where you create, or connect, your workforce identities in AWS once and manage access centrally across your AWS organization.

Internet gateway

The internet gateway allows resources deployed in AWS to communicate to the internet if required.

Application load balancer
The Application Load Balancer integrates with the Auto Scaling groups and will spread user traffic across the available EKS nodes. The Auto scaling groups will scale the EKS nodes depending on the traffic received.

DNS
Route53 DNS records will be configured and direct traffic to the Application Load Balancer which will direct traffic to the EKS nodes hosting the Cars web application.

EKS control plane
The EKS control plane is seen as the master node of the EKS cluster. The EKS cluster manages the EKS nodes.

EKS nodes
The EKS nodes will host the containerized Cars web application frontend. The EKS nodes will connect to a RDS database and the S3 bucket.

Autoscaling groups
The autoscaling group integrates with the EKS cluster and allows for the EKS cluster to scale as user traffic increases.

Container Registry
The Elastic Container Registry will be used as the Container repository where the images for the Cars web application front end will be stored. During application updates, the images will be updated with the CI / CD process. The EKS cluster will automatically replace EKS nodes one by one with the updated Cars web application images. This will allow for the application to be highly available.

DB storage
The Database storage will be hosted on a AWS RDS instance with multi-AZ enabled.

File storage
AWS S3 will be used as the file storage where images will be stored. AWS S3 allows for unlimited storage at a low cost.

CI / CD pipeline
The CI / CD pipeline will be implemented and used to deploy all resources to the AWS account. The CI / CD pipeline will integrate unit testing, code builds and integration testing.

Infrastructure as Code (Terraform)
All resources will be built with Infrastructure as Code using Terraform.

Git (Version and Source Code control)
All application and infrastructure code will be hosted on a private GitHub repository.

Cost and Application Logging & Monitoring
Logging and monitoring will span across two areas. One area is the monitoring of resource costs, another will be the monitoring and logging of the Cars web application. Logging and monitoring metrics will be stored in AWS Cloudwatch. AWS Cost explorer and AWS Budgets will be used for cost monitoring. All logging and monitoring will be integrated with AWS Cloudwatch and AWS SNS for alerts and notifications.

Notifications
SNS topics will be created and allow for notifying all parties involved in the solution on the resource costs as well as the logging and monitoring of the application.

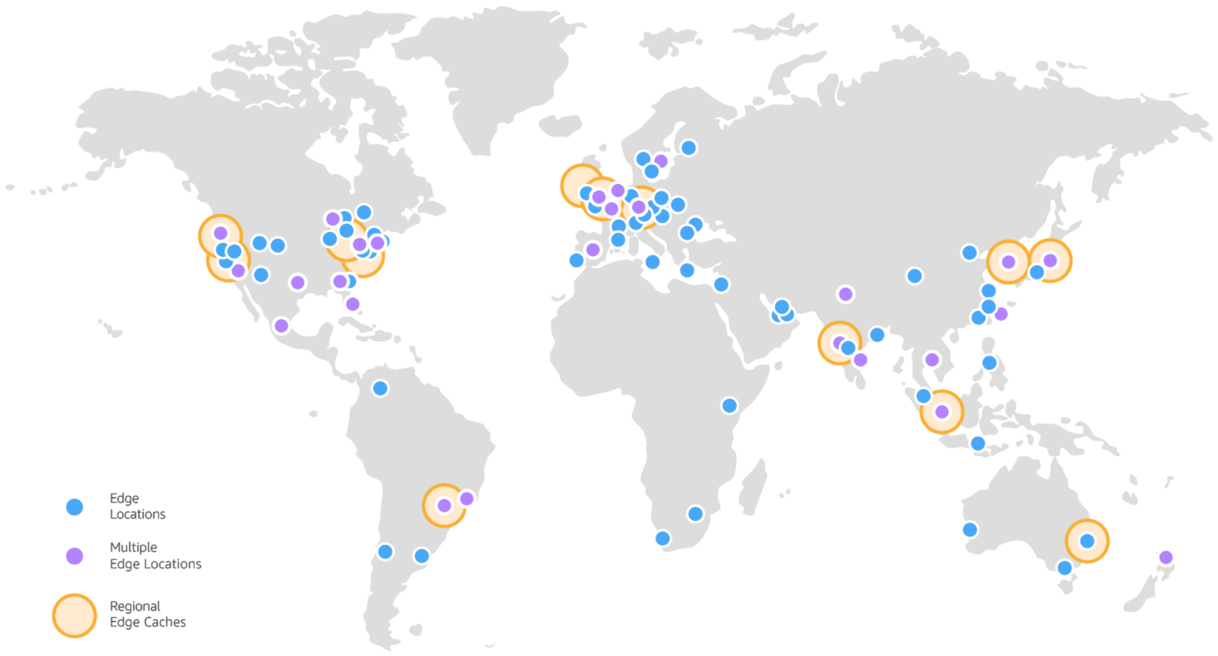
3.4.2 Phase 2 - Global scalability

Below is a detailed description of each resource that will be deployed with the global scalability phase.

CDN

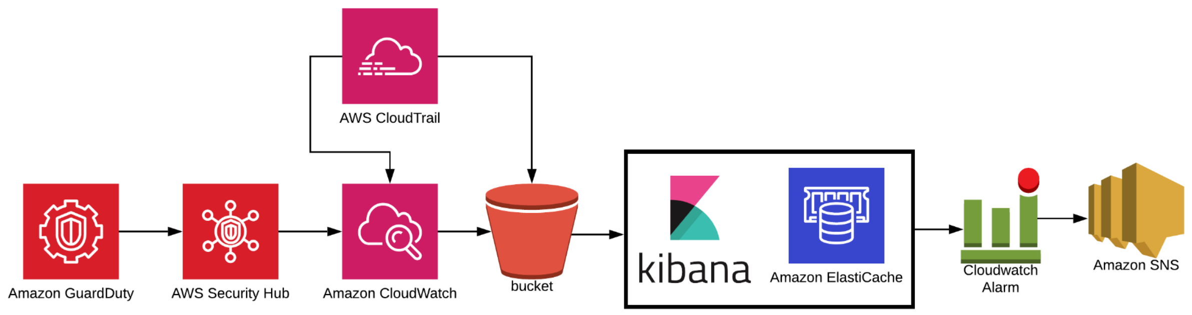
Cloudfront will be used as a CDN. The CDN will provide a globally-distributed network of proxy servers that cache content, thus improving access speed of the Cars web application for users spread across all regions. The CDN will allow for scaling the application access globally as the Cars web application expands globally into other regions such as Asia and the Americas.

AWS Cloudfront currently covers a wide range of regions globally as seen in the diagram below.



3.4.3 Phase 3 - Security improvements

Below is a detailed description of each resource that will be deployed with the security phase 3 improvements phase.



Example design for security monitoring, logging and reporting in an AWS environment.

AWS Security Hub

AWS Security Hub gives you a comprehensive view of your security alerts and security posture across your AWS accounts

Amazon Guard Duty

Amazon GuardDuty is a threat detection service that continuously monitors for malicious activity and unauthorized behavior to protect your AWS accounts, workloads, and data stored in Amazon S3.

AWS WAF

AWS WAF is a web application firewall that helps protect your web applications or APIs against common web exploits and bots that may affect availability, compromise security, or consume excessive resources.

AWS Cognito

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Apple, Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0 and OpenID Connect.

AWS Trusted Advisor

AWS Trusted advisor will provide recommendations and best practices for securing your AWS environment.

AWS Cloudtrail

AWS CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of your AWS account.

Kibana with Elasticache

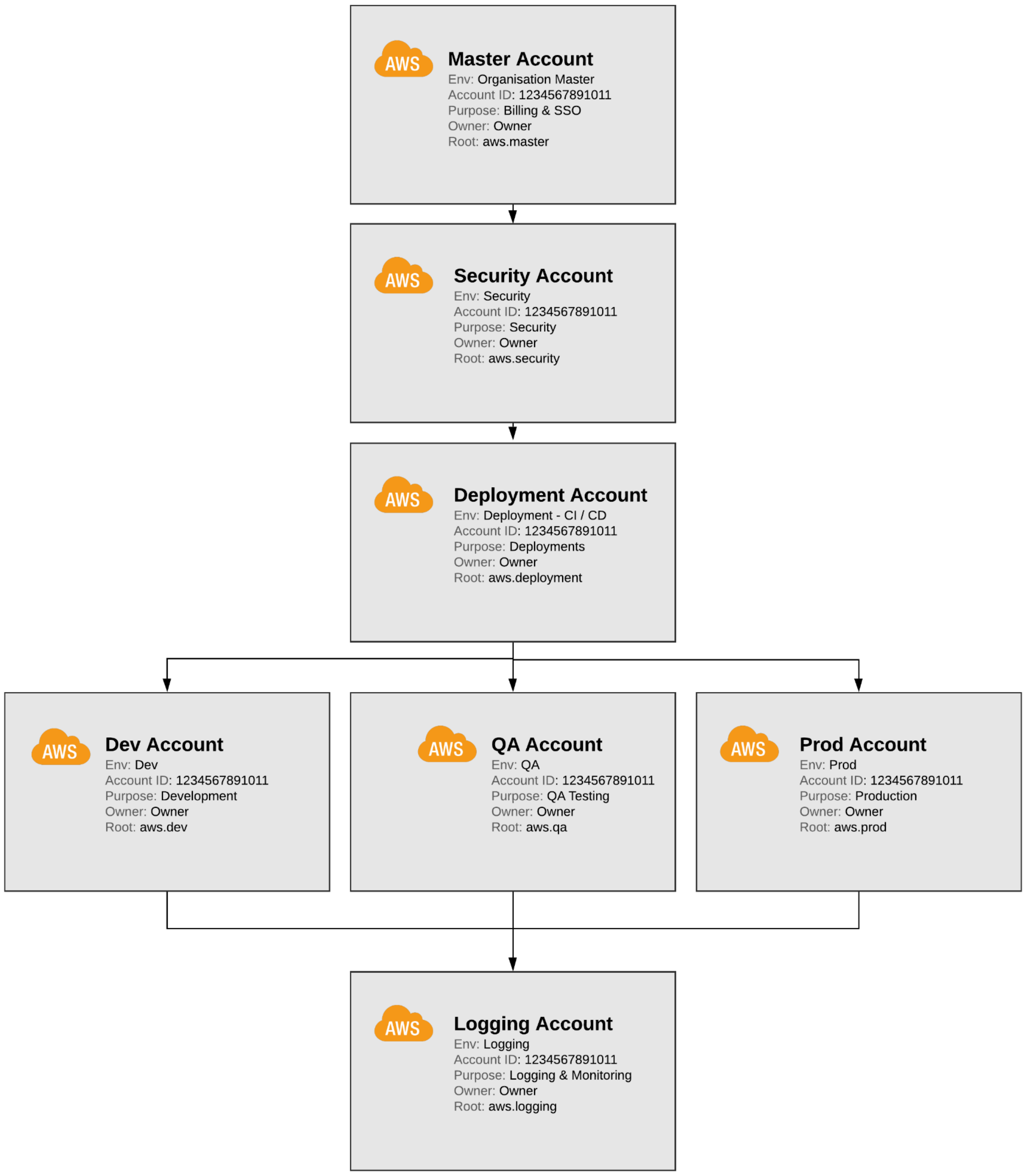
With a Kibana implementation running on Elasticache you can implement a security dashboard.
The following is an example of such an implementation.

https://aws.amazon.com/blogs/security/how-to-visualize-and-refine-your-networks-security-by-adding-security-group-ids-to-your-vpc-flow-logs/

Section 4-Extra Design Features/Outstanding Issues

Multi-Environments Account Architecture

A multi account design can be implemented to allow for separation of services. This can allow for a better level of control around costs and provide a better view of the overall environment.
Link: https://medium.com/@sanchitbansal26/aws-multi-account-architecture-14ce665a5096



Multi-Region Architecture

A multi-regional environment can be implemented to allow for failover / DR in situations where an outage occurs in the primary region of eu-west-1.
Link: https://aws.amazon.com/solutions/implementations/multi-region-application-architecture/

Section 5 – References

https://aws.amazon.com/

https://aws.amazon.com/organizations/

https://aws.amazon.com/eks/features/

https://aws.amazon.com/ecs/features/

https://aws.amazon.com/devops/

https://aws.amazon.com/security/

https://aws.amazon.com/cognito/

https://aws.amazon.com/pricing/

https://www.terraform.io/

https://github.com/

https://aws.amazon.com/blogs/modernizing-with-aws/containerizing-legacy-asp-net-applications-using-aws-app2container-a2c/

Section 6 – Glossary

Glossary of terms / acronyms

VPC	Virtual Private Network
EKS	Elastic Kubernetes Service
ECS	Elastic Container Service
S3	Simple Storage Service
RDS	Relational Database Service
AWS WAF	Web Application Firewall
AWS IAM	Identity and Access Management
SNS	Simple Notification Service
IaC	Infrastructure as Code