<Stargo scenario 1>

# System Design Document

Version 1.0

30/11/2023

תוכן

[System Design Document i](#_Toc152247569)

[1. Introduction 2](#_Toc152247570)

[1.1 Purpose of the SDD 2](#_Toc152247571)

[2. General Overview and Design Guidelines/Approach 3](#_Toc152247572)

[2.1 General Overview 3](#_Toc152247573)

[2.2 Assumptions/Constraints/Risks 3](#_Toc152247574)

[3. System Architecture and Security Design 6](#_Toc152247575)

[3.1 Cloud Security Architecture 8](#_Toc152247576)

[4. CICD 9](#_Toc152247577)

[4.1 Business Requirements 9](#_Toc152247578)

[4.2 Components 9](#_Toc152247579)

## Introduction

In the dynamic landscape of modern software development, deploying a new service is a multifaceted task that demands meticulous planning and strategic implementation. This scenario presents a backend developer with the challenge of deploying a service whose code resides in Bitbucket, intended to run on either an ECS (Elastic Container Service) or EKS (Elastic Kubernetes Service) cluster. Moreover, the service needs to securely connect to a private database, both residing within the same Virtual Private Cloud (VPC). The intricate interplay of these components necessitates a well-thought-out network design with robust security measures.

In addition to the backend service, there is a requirement to expose a static website hosted on Amazon S3 with HTTPS to the public. This website is designed to showcase a list of movies sourced from an RDS (Relational Database Service). The website, in turn, needs to communicate with the aforementioned backend service to pull data from the private database. This intricate web of services and interactions demands a comprehensive understanding of both network architecture and the Continuous Integration/Continuous Deployment (CI/CD) process.

### Purpose of the SDD

This SSD (System Specification Document) aims to provide a detailed insight into the design of the network infrastructure, the stages involved in the CI/CD process for the backend service, and the intricacies of exposing a secure static website to the public. Let's delve into each aspect to construct a robust and scalable solution that aligns with industry best practices.

## General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

### General Overview

The system at hand represents a sophisticated deployment scenario for a new backend service, seamlessly integrating with an ECS or EKS cluster. This deployment is complemented by a private database coexisting within the same VPC. The overarching goal is to establish a secure, scalable, and efficient network architecture that facilitates the secure interaction of these components. Additionally, the requirement to expose a static website hosted on Amazon S3 to the public adds a layer of complexity, demanding careful orchestration of communication between the website, backend service, and the underlying database.

### Assumptions/Constraints/Risks

#### Assumptions

1. **AWS Environment:** It is assumed that the system is deployed within the Amazon Web Services (AWS) environment, leveraging services such as ECS, EKS, S3, and RDS.
2. **VPC Existence:** The Virtual Private Cloud (VPC) is assumed to be in place, facilitating the secure co-location of the ECS or EKS cluster, the private database (RDS), and other necessary components.
3. **Containerization:** The backend service is containerized, either using ECS or EKS, allowing for efficient deployment, scaling, and management.
4. **Security Measures:** Security best practices, including encryption in transit and at rest, IAM roles, and restrictive access controls, are assumed to be implemented to safeguard sensitive data.
5. **Static Website Hosting:** The static website is hosted on Amazon S3, ensuring simplicity in web hosting and content delivery.
6. **Continuous Integration/Continuous Deployment (CI/CD):** A CI/CD pipeline is assumed to be in place for automated testing, building, and deploying the backend service updates.
7. **Website Communication:** The static website securely communicates with the backend service to retrieve movie data from the private database.
8. **Network Isolation:** The VPC ensures network isolation and controlled communication among the backend service, database, and other components for enhanced security.

These assumptions provide a foundational understanding for the system's design and are integral to the successful deployment and operation of the described architecture.

#### Constraints

1. **AWS Dependency:** The system relies on Amazon Web Services (AWS) for its infrastructure, which may limit flexibility for migration to other cloud providers.
2. **Service Downtime:** Performing updates or maintenance on the ECS/EKS cluster or backend service may result in temporary service downtime, impacting user experience.
3. **Cost Considerations:** AWS services entail costs, and careful monitoring and optimization are required to manage expenses effectively, especially in dynamic or resource-intensive applications.
4. **Skill Dependencies:** Effective implementation and management of the system require skilled AWS professionals familiar with ECS, EKS, S3, and associated services.
5. **Regulatory Compliance:** Depending on the nature of the data stored in the private database, regulatory compliance requirements may impose constraints on data handling and storage practices.

These constraints highlight the need for careful consideration and strategic planning to navigate potential challenges and ensure the smooth operation of the system within the defined parameters.

#### Risks

1. Security Risks:
   * Data Breach: Inadequate security measures could lead to unauthorized access to sensitive data stored in the private database, posing a risk of data breaches.
   * Network Vulnerabilities: Security loopholes in the VPC or misconfigurations might expose the system to external threats, necessitating vigilant monitoring and regular security audits.
   * Container Security: Vulnerabilities within container images or runtime environments could compromise the integrity of the backend service, emphasizing the need for robust container security practices.
2. CI/CD Focus:
   * Pipeline Vulnerabilities: Flaws in the CI/CD pipeline might introduce security vulnerabilities into the deployed application. Ensuring secure coding practices and thorough testing at each stage of the pipeline is essential.
   * Automated Testing: Incomplete or ineffective automated testing could result in the deployment of faulty updates, emphasizing the importance of comprehensive test coverage within the CI/CD process.
   * Version Control: Strict version control practices are crucial to avoid issues related to code divergence and ensure a reliable and traceable deployment process.
3. Downtime Concerns:
   * Routine Maintenance: Planned maintenance activities on the ECS/EKS cluster or backend service could lead to temporary downtime. Strategies for minimizing the impact on users during scheduled maintenance must be implemented.
   * Unplanned Outages: Unforeseen issues, such as infrastructure failures or sudden increases in user traffic, may lead to service outages. Implementing redundancy and failover mechanisms is critical to mitigate such risks.
   * Rollback Procedures: In the event of deployment failures, having efficient rollback procedures in place is essential to quickly revert to a stable state and minimize downtime.
4. Mitigation Strategies:
   * Security Audits: Conduct regular security audits to identify and address vulnerabilities in the system. Implement encryption, secure networking, and robust IAM policies to enhance overall security.
   * Automated Security Scanning: Integrate automated security scanning tools into the CI/CD pipeline to identify and rectify security issues early in the development process.
   * Continuous Monitoring: Employ continuous monitoring tools to detect and respond to security incidents promptly, reducing the risk of prolonged security breaches.
   * Redundancy and Load Balancing: Implement redundancy and load balancing mechanisms to enhance system availability and minimize the impact of potential downtime.
   * Rolling Deployments: Adopt rolling deployment strategies to minimize downtime during updates, ensuring a seamless user experience while maintaining service availability.

These risk considerations and focused mitigation strategies aim to address key aspects of security, CI/CD, and downtime, safeguarding the integrity and reliability of the deployed system.

## System Architecture and Security Design

The system is architecturally designed using a multi-tier approach, leveraging cloud services and Kubernetes to ensure scalability, maintainability, and security. The primary responsibilities are distributed across six major components: the Static Web Hosting on Amazon S3, the Backend Service deployed on Kubernetes (EKS),Conectivity to the web via CloudFront,Conectivity to the Backend Via API gateway,Data store in RDS, and the Continuous Integration/Continuous Deployment (CI/CD) pipeline managed by Jenkins.

1. Static Web Hosting (Amazon S3):
   * Responsibilities:
     + Hosting static website content.
     + Serving assets such as HTML, CSS, and JavaScript.
   * Architecture:
     + Utilizes Amazon S3 for scalable and cost-effective static web hosting.
     + Amazon CloudFront is employed as a content delivery network (CDN) for HTTPS support and efficient content distribution.
2. Backend Service (Kubernetes – EKS,API Gateway):
   * Responsibilities:
     + Executing core application logic.
     + Connecting securely to the private database.
   * Architecture:
     + Cloudfront accesses the api gateway which accesses EKS
     + Containerized application deployed on Kubernetes (EKS).
     + Implements a multi-tier architecture with separate layers for presentation, business logic, and data access.
3. Continuous Integration/Continuous Deployment (CI/CD - Jenkins on Kubernetes):
   * Responsibilities:
     + Automating testing, building, and deploying the application.
   * Architecture:
     + Jenkins is orchestrated within a Kubernetes pod to ensure scalability and ease of management.
     + CI/CD pipeline triggers automatic deployment of updates to both S3 and EKS based on Git repository changes.

Interaction and Data Flow:

1. Static Web Hosting to End Users:
   * CloudFront securely delivers static assets from S3 to end users via HTTPS, enhancing content delivery and user experience.
   * CloudFront securely delivers dynamic assets from API gateway to end users via HTTPS, enhancing content delivery and user experience.
2. Backend Service Interaction:
   * The static website hosted on S3 behind CloudFront ,CloudFront communicates with the backend service (EKS) over HTTPS via API gateway for dynamic content.
   * Secure communication ensures data integrity and confidentiality between the website and backend.
3. CI/CD Pipeline Execution:
   * Jenkins, orchestrated within a Kubernetes pod, executes the CI/CD pipeline.
   * Jenkins communicates with the Git repository to detect changes and triggers automated testing and deployment processes.

Rationale for Decomposition:

* Multi-Tier Architecture: This architectural choice aligns with best practices, separating concerns into distinct layers, facilitating maintainability, and allowing for scalability as each tier can scale independently.
* CloudFront for HTTPS: CloudFront provides HTTPS support, enhancing security and performance for content delivery.
* Kubernetes for Container Orchestration: Kubernetes (EKS) allows for efficient container orchestration, scaling, and management, aligning with modern deployment practices.
* Jenkins for CI/CD: Jenkins orchestrates the CI/CD pipeline, automating testing and deployment processes, promoting rapid and reliable releases.

Security Considerations:

* IAM Roles and Permissions: IAM roles are assigned to each component with least-privileged permissions, ensuring secure access to AWS services.
* Security Groups: Components are associated with specific security groups, enforcing network security and restricting communication based on defined patterns.
* Encryption: HTTPS is enforced for data in transit, both between CloudFront and users and between the static website and the backend service.
* Common Security Patterns: Follows common security patterns such as the principle of least privilege, secure network configurations, and encryption to mitigate potential vulnerabilities.

Conclusion: This architectural overview establishes a scalable, secure, and maintainable system, incorporating key cloud services, Kubernetes orchestration, and CI/CD automation. The design choices aim to optimize each component's responsibilities, fostering collaboration to achieve the desired functionality with an emphasis on security and efficiency.

This section outlines the system and hardware architecture design of the system.

### Cloud Security Architecture

#### Security Groups

###### Static Web Hosting (CloudFront):

Security Group Name: sg-static-web-hosting

* Inbound Rules:
  1. Allow Inbound HTTPS (Port 443) traffic from 0.0.0.0/0 for users accessing the website.
  2. Allow Inbound HTTP (Port 80) traffic, redirecting to HTTPS, if needed.
  3. No other inbound traffic allowed.
* Outbound Rules:
  1. Allow all outbound traffic to ensure S3 and CloudFront functionality.

###### Backend Service (Kubernetes - EKS):

Security Group Name: sg-backend-service

* Inbound Rules:
  1. Allow Inbound HTTPS (Port 443) traffic from the security group of the Jenkins pod.
  2. Allow Inbound traffic from the private subnet of the VPC for communication with the private database.
  3. Deny all other inbound traffic.
* Outbound Rules:
  1. Allow all outbound traffic for the backend service to interact with external services and databases.

###### Continuous Integration/Continuous Deployment (Jenkins on Kubernetes):

Security Group Name: sg-jenkins

* Inbound Rules:
  + Allow Inbound SSH (Port 22) traffic from the developer's IP address for secure access.
  + Allow Inbound HTTPS (Port 443) traffic from the Git repository for CI/CD pipeline trigger.
  + Allow Inbound traffic from the security group of the backend service for deploying updates.
  + Deny all other inbound traffic.
* Outbound Rules:
  + Allow all outbound traffic for Jenkins to communicate with external services and the Git repository.

###### Private Database (RDS):

Security Group Name: sg-private-database

* Inbound Rules:
  + Allow Inbound traffic from the security group of the backend service for database interaction.
  + Deny all other inbound traffic.
* Outbound Rules:
  + Allow all outbound traffic to enable the database to communicate with the backend service.

###### EKS Worker Nodes (Associated with Backend Service)

Security Group Name: sg-eks-worker-nodes

* Inbound Rules:
  + Allow Inbound traffic from the security group of the backend service for communication.
  + Deny all other inbound traffic.
* Outbound Rules:
  + Allow all outbound traffic to facilitate interactions between worker nodes and external services.

#### CloudFront Security

###### ACM Certificate for CloudFront

To secure communication over HTTPS between CloudFront and end-users, an AWS Certificate Manager (ACM) certificate is utilized. This certificate ensures the encryption of data in transit, enhancing the security of the static web hosting on Amazon S3.

* **Certificate Name:** cloudfront-acm-certificate
* **Domain Name:** stargo.com
* **Certificate Type:**
  + **Public Certificate:** As the website is public-facing, a public certificate is used to secure connections over the internet.
* **Validation Method:**
  + **DNS Validation:** ACM validates ownership by adding a DNS record to the domain's DNS configuration.
* **Subject Alternative Names (SANs):**
  + \*.stargo.com (Wildcard certificate for subdomains, if applicable)
* **Key Algorithm:**
  + RSA 2048-bit (Standard key size for compatibility with most clients)
* **Security:**
  + **Managed by ACM:** The certificate is managed by ACM, ensuring automatic renewal and reducing administrative overhead.
* **Integration with CloudFront:**
  + The ACM certificate is associated with the CloudFront distribution to enable secure HTTPS communication.
* **Renewal:**
  + ACM automatically handles certificate renewal, relieving the need for manual intervention.

###### WAF

Set Up AWS WAF:

* **Create WebACL:**
  + In the AWS WAF console, create a WebACL (Web Access Control List) to define the rules and conditions for filtering requests.
* **Define Rules:**
  + Define rules within the WebACL to identify and block malicious traffic based on specific conditions (e.g., SQL injection, XSS attacks).

Associate WAF with CloudFront Distribution:

* **Associate WebACL:**
  + In the CloudFront console, associate the created WebACL with your CloudFront distribution.
* **Specify WAF Rules:**
  + Specify the WAF rules and conditions that should be applied to incoming requests served by CloudFront.

Rate Limiting and IP Blocking:

* **Rate Limiting:**
  + Utilize WAF rate limiting rules to control the rate at which requests are allowed, mitigating against potential DDoS attacks.
* **IP Blocking:**
  + Configure IP sets within WAF to block specific IP addresses or IP ranges associated with malicious activity.

Custom Rules and Regex Patterns:

* **Custom Rules:**
  + Create custom rules based on your application's specific security requirements.
* **Regex Patterns:**
  + Leverage regex patterns in WAF rules for more advanced pattern matching and protection.

###### Authentication

Set Up AWS Cognito

* Create a User Pool:
  + In the Amazon Cognito console, create a user pool to manage user identities and authentication.
* Configure App Clients:
  + Create an app client within the user pool to represent your application.

Enable AWS Cognito Authentication:

* **Configure CloudFront Behaviors:**
  + Configure the CloudFront behavior settings to use your Cognito user pool for authentication.
  + Under the "Behaviors" settings, choose the behavior you want to protect and select "Viewer Protocol Policy" as "Redirect HTTP to HTTPS" for secure communication.
  + Under "Lambda Function Associations," you may configure a Lambda@Edge function for additional customization if needed.

Set Up Authorization in Cognito:

* **Define App Client Settings:**
  + In your Cognito user pool, define the allowed OAuth 2.0 flows, scopes, and set up the client secret.
* **Define Domain Name:**
  + Optionally, set up a custom domain for your Cognito user pool for a more branded user experience.

:

#### Subnets

###### Public Subnet (Static Web Hosting - S3 and CloudFront):

* **Route Table Name:** rt-public-subnet-static-web
  + **Associated Subnet:** Public subnet containing resources for static web hosting.
  + **Routes:**
    - 0.0.0.0/0 to Internet Gateway (IGW): Allows outbound internet access for resources in the public subnet.
    - Local VPC route: Enables local communication within the VPC.

###### Private Subnet (Backend Service - EKS):

* **Route Table Name:** rt-private-subnet-backend-service
  + **Associated Subnet:** Private subnet containing resources for the backend service hosted on EKS.
  + **Routes:**
    - 0.0.0.0/0 to NAT Gateway or NAT Instance: Allows outbound internet access for resources in the private subnet.
    - Local VPC route: Enables local communication within the VPC.
    - Routes to specific services, such as RDS or other internal services, depending on the architecture.

###### Private Subnet (Continuous Integration/Continuous Deployment - Jenkins):

* **Route Table Name:** rt-private-subnet-jenkins
  + **Associated Subnet:** Private subnet containing resources for Jenkins on Kubernetes.
  + **Routes:**
    - 0.0.0.0/0 to NAT Gateway or NAT Instance: Allows outbound internet access for resources in the private subnet.
    - Local VPC route: Enables local communication within the VPC.
    - Routes to specific services needed for CI/CD, such as access to ECR, S3, and other relevant AWS services.

###### Private Subnet (Private Database - RDS):

* **Route Table Name:** rt-private-subnet-database
  + **Associated Subnet:** Private subnet containing resources for the private database (RDS).
  + **Routes:**
    - 0.0.0.0/0 to NAT Gateway or NAT Instance: Allows outbound internet access for resources in the private subnet (if needed).
    - Local VPC route: Enables local communication within the VPC.
    - Specific routes for communication with the backend service and other relevant services.

###### Private Subnet (EKS Worker Nodes):

* **Route Table Name:** rt-private-subnet-eks-worker-nodes
  + **Associated Subnet:** Private subnet containing resources for EKS worker nodes.
  + **Routes:**
    - 0.0.0.0/0 to NAT Gateway or NAT Instance: Allows outbound internet access for resources in the private subnet.
    - Local VPC route: Enables local communication within the VPC.
    - Routes specific to EKS cluster communication and access to relevant AWS services.

## CICD

### Business Requirements

Continuous Integration and Continuous Deployment (CI/CD) play a crucial role in modern software development practices, streamlining the process of delivering software updates and improvements. The business requirements for implementing CI/CD typically revolve around improving development efficiency, ensuring software quality, and accelerating time-to-market.

### Components

#### Source Code Repository

The source code repository selected for the project is bitbucket as it integrates well with existing tools in the organization such as Jira as both are made by Attlassian

#### Version Control System (VCS)

The VCS selected is git as it is an industry standard and most industry professionals already are familiar with it

#### Build Server

The build server select is Jenking due to its many plugins which integrate well with existing architectures in the organization.

#### Build Script

` the build scripts is a Jenkins file which is responsible for the CICD process end to end

Bellow is a description of the process:

1. Webhook Trigger:
   * This stage is triggered by a webhook event, such as a push to the version control system.
2. Checkout Code:
   * Checks out the source code from bitbucket.
3. Build Backend:
   * Builds the backend application using Gradle(if app is in java).
4. Unit Tests:
   * Runs unit tests for the backend application with Selenium and run vulnerability checks with SonarQube.
5. Build Docker Image:
   * Builds a Docker image for the backend application.
6. Push Docker Image to ECR:
   * Pushes the Docker image to Amazon Elastic Container Registry (ECR).
7. Deploy Backend to EKS:
   * Deploys the backend application to Kubernetes (EKS). Ensure Kubernetes configurations are set up appropriately and “imagePull” is set to “Always”.
8. Upload Static Website to S3:
   * Uploads the static website to Amazon S3.

#### Artifact Repository

The artifact repository chosen is ECR as it easy and simple to setup,secures your images in the cloud and not externally and is managed by AWS so it also reduces the operationial overhead of managing a registry like nexus for example..

1. to the author before finalizing the initial draft of the document.