**Modern Serverless Application with AWS**

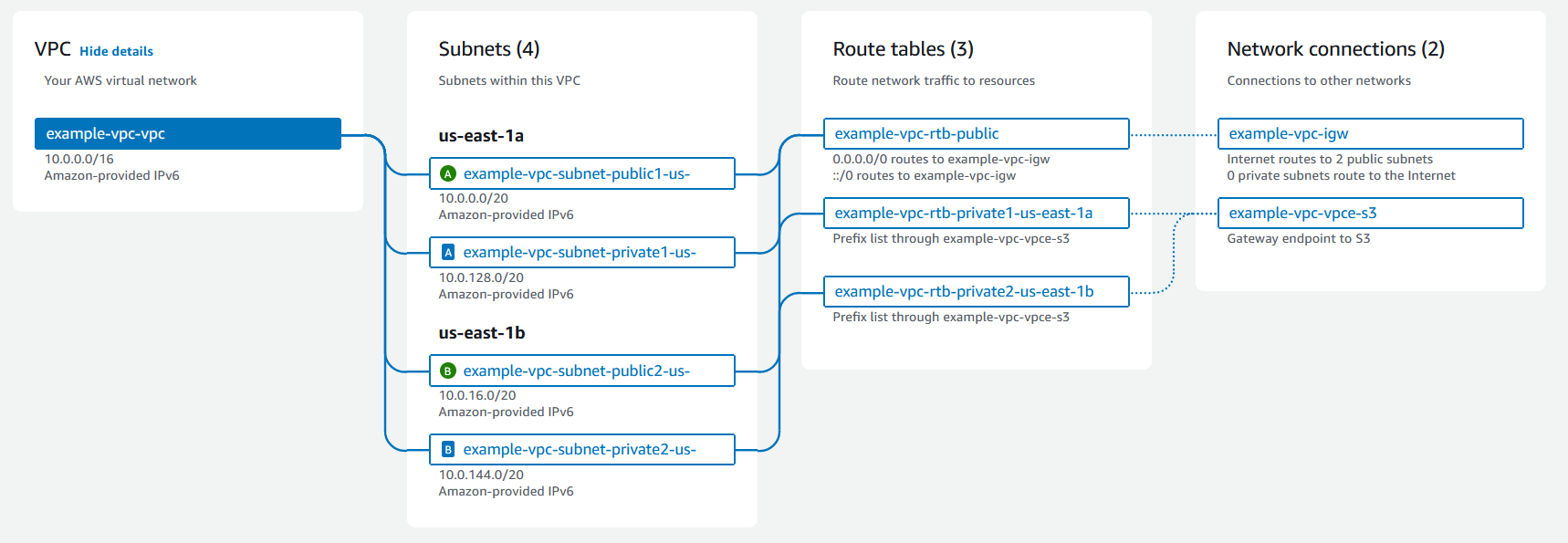
1. **Architecture Diagram:** Create a detailed and well-labeled diagram illustrating the

proposed architecture. The diagram should clearly show

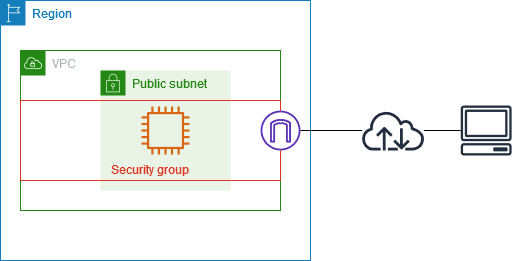
security groups.

With Amazon Virtual Private Cloud (Amazon VPC), you can launch AWS resources in a logically isolated virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

The following diagram shows an example VPC. The VPC has one subnet in each of the Availability Zones in the Region, EC2 instances in each subnet, and an internet gateway to allow communication between the resources in your VPC and the internet.



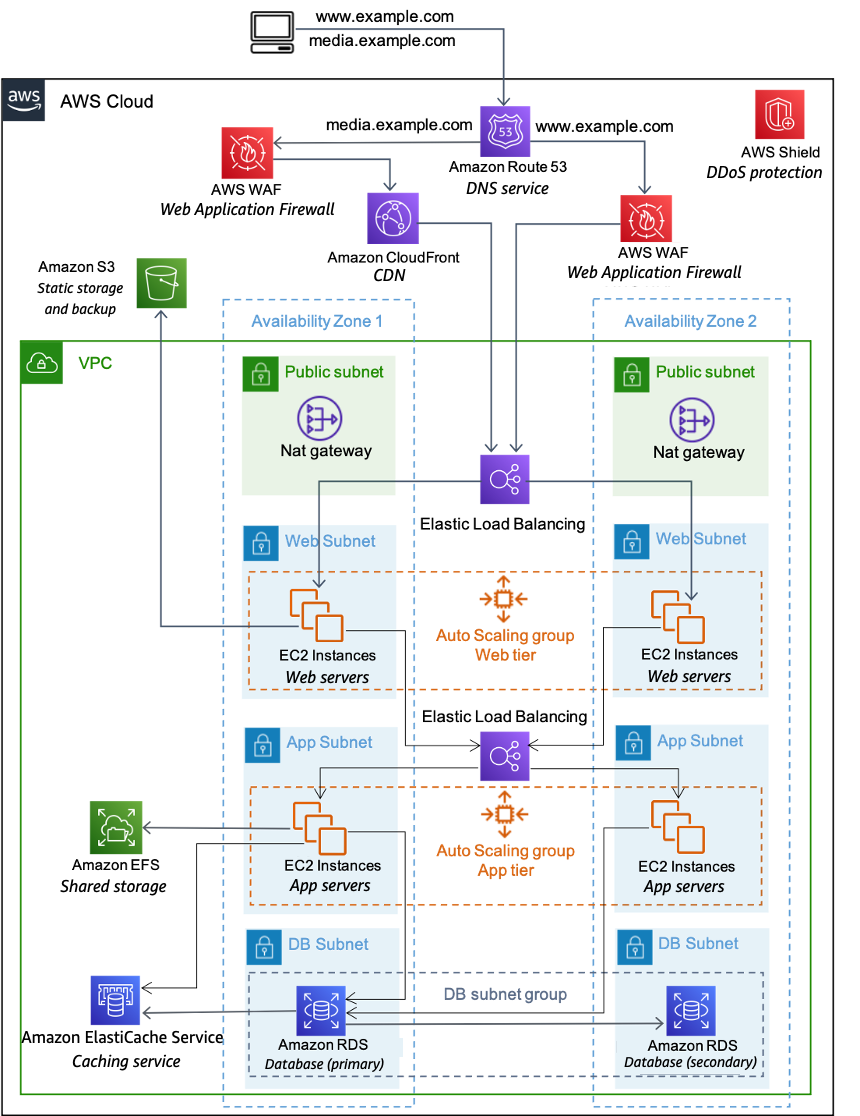
Security Groups (IAM):



* **Components:** The main components of the system (web app, mobile app,

backend services, databases, etc.) and their interactions.

The following figure provides another look at that classic web application architecture and how it can leverage the AWS Cloud computing infrastructure.



Elastic Beanstalk is the simplest way to deploy and run your web application on Amazon Web Services. Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, automatic scaling, and web application health monitoring.

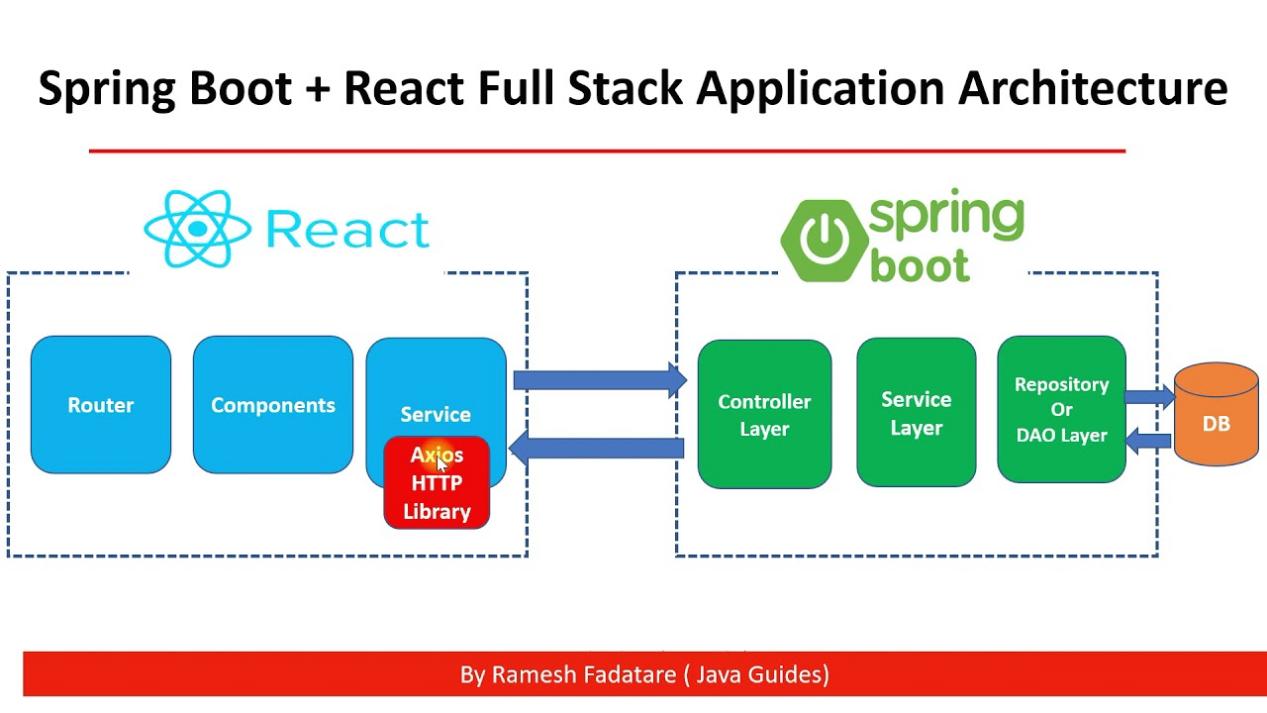
We can run web application in Full Stack mode with ReactJS as frontend and Python as backend using Amazon Elastic Beanstalk container service. Backend service (Python) shall connect to database such as Amazon RDS.

* **Data Flows:** How data moves between components, including data ingestion,

processing, storage, and retrieval.

As shown in Full Stack application architecture, users connect to frontend (ReactJS) as static web service, data flow from Front End to Back End (Java Spring Boot) and vice versa via the URL of Back End. Finally Back End communicate to the database (Amazon RDS).

Data ingestion at Front End from users then process at Back End as API services. Finally data is stored and retrieved at database service.



1. **Component Design (High-Level):** Describe the responsibilities of each major

component:

* **Web/Mobile App:** Outline the frontend architecture and how it communicates

with the backend.

The frontend elements structure of a website are defined by HTML, styled by CSS to produce an eye-catching visual, and integrated with **JavaScript** to add interaction and general functionality. Additionally, some frameworks like [React](https://www.geeksforgeeks.org/reactjs-tutorials/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank), [Vue](https://www.geeksforgeeks.org/vue-js/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank), or [Angular](https://www.geeksforgeeks.org/introduction-to-angularjs/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank) make development easier and faster by providing specific environment components.

In backend development use languages like Python, Java, Ruby, PHP, and JavaScript to build and manage server-side software. This phase manages requests from the frontend, interacts with databases and APIs, and then ensures data storage

## **Communications between frontend and backend using 3 methods:**

### 1. RESTful APIs:

[REST (Representational State Transfer)](https://www.geeksforgeeks.org/rest-api-introduction/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank) is an architectural style for creating web services.  This is the most popular approach. It generally uses HTTP request and response methods in order to exchange data in a normalize format. The backend exposes different endpoints for multiple functionalities, and then frontend makes calls to these endpoints in order to retrieve or manipulate data (API).

### 2. WebSockets:

A persistent, bi-directional communication protocol that connects a client and a server is called[WebSockets.](https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank) WebSockets, in contrast to conventional HTTP, allow for continuous communication, which makes them appropriate for applications that need real-time updates.

### 3. Server-Side Rendering (SSR):

In Server-Side Rendering, the server crafts the webpage’s HTML and sends it to the browser, sparing the client’s browser from this hefty task. The initial page loads much more quickly with this technique, which also improves [search engine optimisation](https://www.geeksforgeeks.org/search-engine-optimization-seo-basics/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank) (**SEO**) and makes it easier for search engines to understand the content.

**Backend Services:** Describe the functions and responsibilities of the serverless

backend services (using AWS Lambda or Fargate).

# AWS Fargate for Amazon ECS

AWS Fargate is a technology that you can use with Amazon ECS to run [containers](https://aws.amazon.com/what-are-containers" \t "https://docs.aws.amazon.com/AmazonECS/latest/developerguide/_blank) without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

When you run your tasks and services with the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task. You configure your task definitions for Fargate by setting the requiresCompatibilities task definition para

meter to FARGATE. For more information, see [Launch types](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definition_parameters.html" \l "requires_compatibilities).

* **Event-Driven Components:** Explain how events are used to trigger actions and

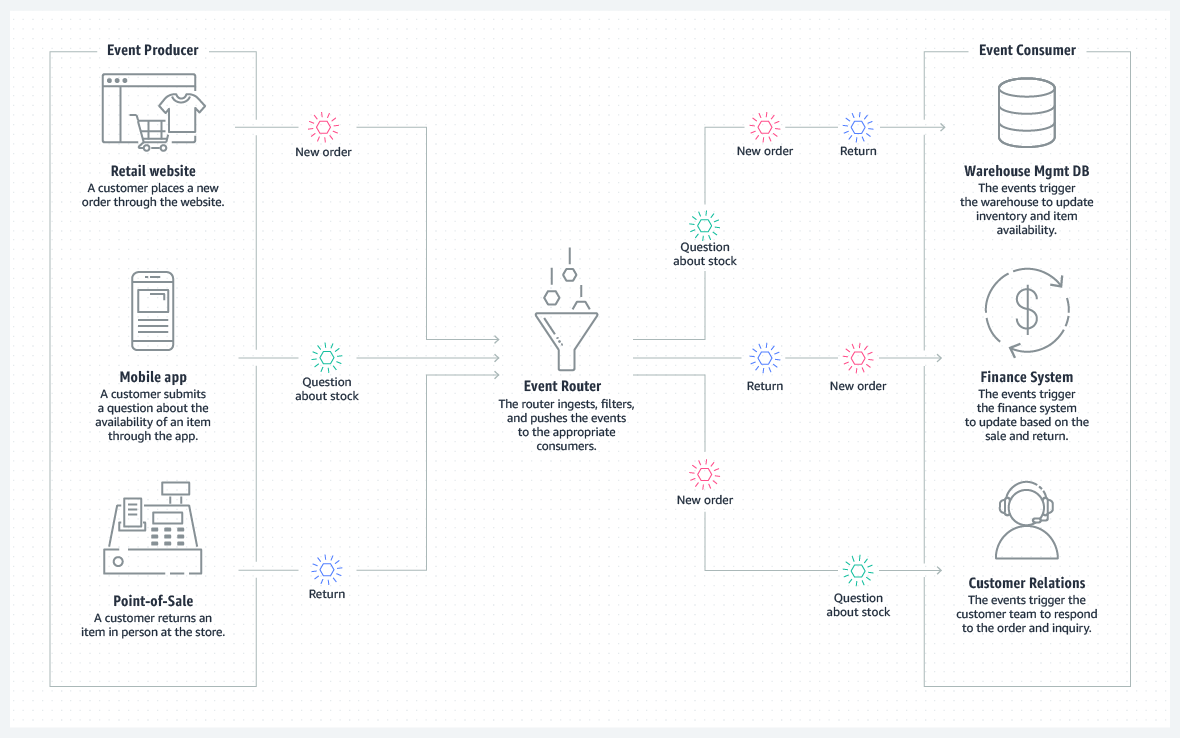
communicate between components.

Amazon Event-Driven Architecture

<https://aws.amazon.com/event-driven-architecture/>

An event-driven architecture uses events to trigger and communicate between decoupled services and is common in modern applications built with microservices. An event is a change in state, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).

Event-driven architectures have three key components: event producers, event routers, and event consumers. A producer publishes an event to the router, which filters and pushes the events to consumers. Producer services and consumer services are decoupled, which allows them to be scaled, updated, and deployed independently.



Confluent - Apache Kafka

Apache Kafka consists of a storage layer and a compute layer that combines efficient, real-time data ingestion, streaming data pipelines, and storage across distributed systems. In short, this enables simplified, data streaming between Kafka and external systems, so you can easily manage real-time data and scale within any type of infrastructure.

* **Data Storage:** Describe the types of data stored in each database (RDS,

DynamoDB, Elasticsearch) and the rationale for using each.

RDS:

* + Amazone Aurora:

Amazon Aurora (Aurora) is a fully managed relational database engine that's compatible with MySQL and PostgreSQL.

DynamoDB:

* Amazon DynamoDB is a serverless, NoSQL, fully managed database with single-digit millisecond performance at any scale.

Elasticsearch:

* Elasticsearch is a distributed, RESTful search and analytics engine capable of addressing a growing number of use cases. As the heart of the Elastic Stack, it centrally stores your data for lightning fast search, fine‑tuned relevancy, and powerful analytics that scale with ease.

* **API:** Explain the API structure (REST or GraphQL) and how it facilitates

communication between the frontend and backend.

### 1. RESTful APIs:

[REST (Representational State Transfer)](https://www.geeksforgeeks.org/rest-api-introduction/" \t "https://www.geeksforgeeks.org/how-to-connect-front-end-and-backend/_blank) is an architectural style for creating web services.  This is the most popular approach. It generally uses HTTP request and response methods in order to exchange data in a normalize format. The backend exposes different endpoints for multiple functionalities, and then frontend makes calls to these endpoints in order to retrieve or manipulate data.

### Procedure:

1. **Client (Frontend)**:
   1. Makes an HTTP request to a specific API endpoint (URL) on the server.
   2. Specifies the request method (GET, POST, PUT, DELETE) and the desired action.
   3. May include request body with data for specific actions like creation or update.
2. **Server (Backend):**
   1. Receives the request and identifies the targeted endpoint based on the URL and method.
   2. Processes the request, accessing databases, performing calculations, or interacting with other services.
   3. Prepares a response containing the requested data, status code (e.g., 200 for success), and any additional information.

**3.** **Security Considerations:** Address security aspects such as:

○ **Authentication and Authorization:** How user access will be managed and

secured.

User authentication and authorization will be managed managed ans secured by Identity and Access Management (IAM).

https://aws.amazon.com/iam/?gclid=EAIaIQobChMI\_8bQudK6hwMVbBiDAx0CUwBREAAYASAAEgKMnfD\_BwE&trk=ed6b9778-818b-45b5-bb8f-a4cd592d7f09&sc\_channel=ps&ef\_id=EAIaIQobChMI\_8bQudK6hwMVbBiDAx0CUwBREAAYASAAEgKMnfD\_BwE:G:s&s\_kwcid=AL!4422!3!651510255315!p!!g!!iam!19835791352!147106034796

○ **Data Protection:** Strategies for encrypting data at rest and in transit.

Data at-rest and data in-transit encryption, AES, and Rivest-Shamir-Adleman (RSA) are the most common and trusted encryption techniques. AES is used to encrypt in-transit and at-rest data, while RSA is typically used for transmitting data between two endpoints.

○ **Network Security:** How the VPC, subnets, and security groups will be

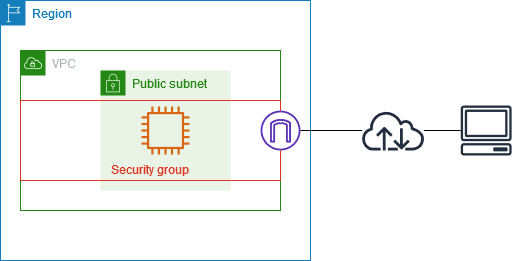
configured to protect resources.

A security group controls the traffic that is allowed to reach and leave the resources that it is associated with. For example, after you associate a security group with an EC2 instance, it controls the inbound and outbound traffic for the instance.

When you create a VPC, it comes with a default security group. You can create additional security groups for a VPC, each with their own inbound and outbound rules. You can specify the source, port range, and protocol for each inbound rule. You can specify the destination, port range, and protocol for each outbound rule.

The following diagram shows a VPC with a subnet, an internet gateway, and a security group. The subnet contains an EC2 instance. The security group is assigned to the instance. The security group acts as a virtual firewall. The only traffic that reaches the instance is the traffic allowed by the security group rules. For example, if the security group contains a rule that allows ICMP traffic to the instance from your network, then you could ping the instance from your computer. If the security group does not contain a rule that allows SSH traffic, then you could not connect to your instance using SSH.

<https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html>

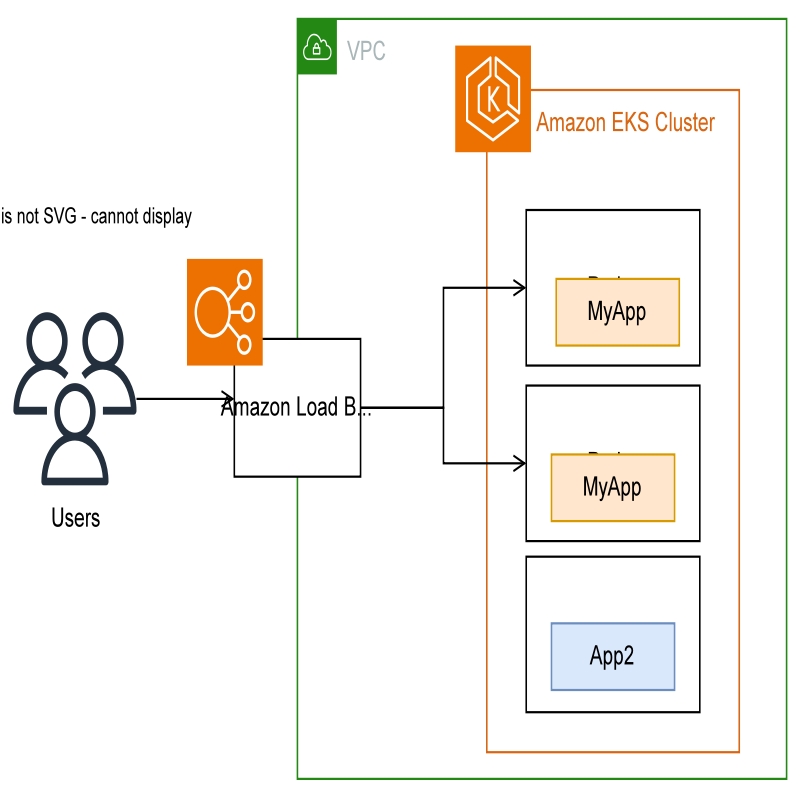


**4.** **Scalability and High Availability:** Explain how the architecture can:

○ Scale horizontally to handle increased traffic or workload.

AWS Load Balancer Controller

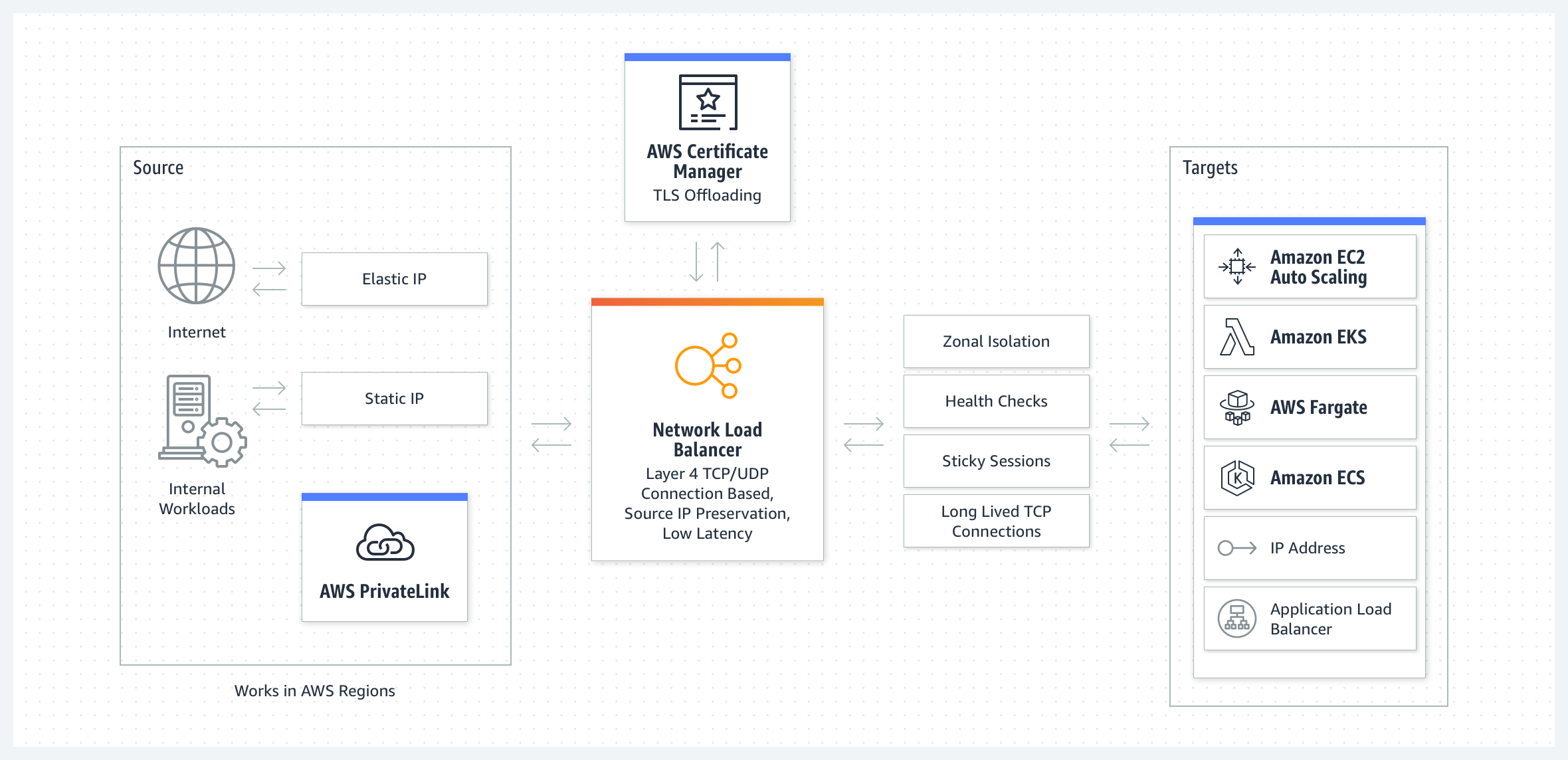
The AWS Load Balancer Controller manages AWS Elastic Load Balancers for a Kubernetes cluster. You can use the controller to expose your cluster apps to the internet. The controller provisions AWS load balancers that point to cluster Service or Ingress resources. In other words, the controller creates a single IP address or DNS name that points to multiple pods in your cluster.



○ Ensure high availability and resilience to failures.

# Route TCP and UDP traffic with Network Load Balancers

Network traffic is load balanced at L4 of the OSI model. To load balance application traffic at L7, you deploy a Kubernetes ingress, which provisions an AWS Application Load Balancer. For more information, see [Route application and HTTP traffic with Application Load Balancers](https://docs.aws.amazon.com/eks/latest/userguide/alb-ingress.html). To learn more about the differences between the two types of load balancing, see [Elastic Load Balancing features](https://aws.amazon.com/elasticloadbalancing/features/" \t "https://docs.aws.amazon.com/eks/latest/userguide/_blank) on the AWS website.



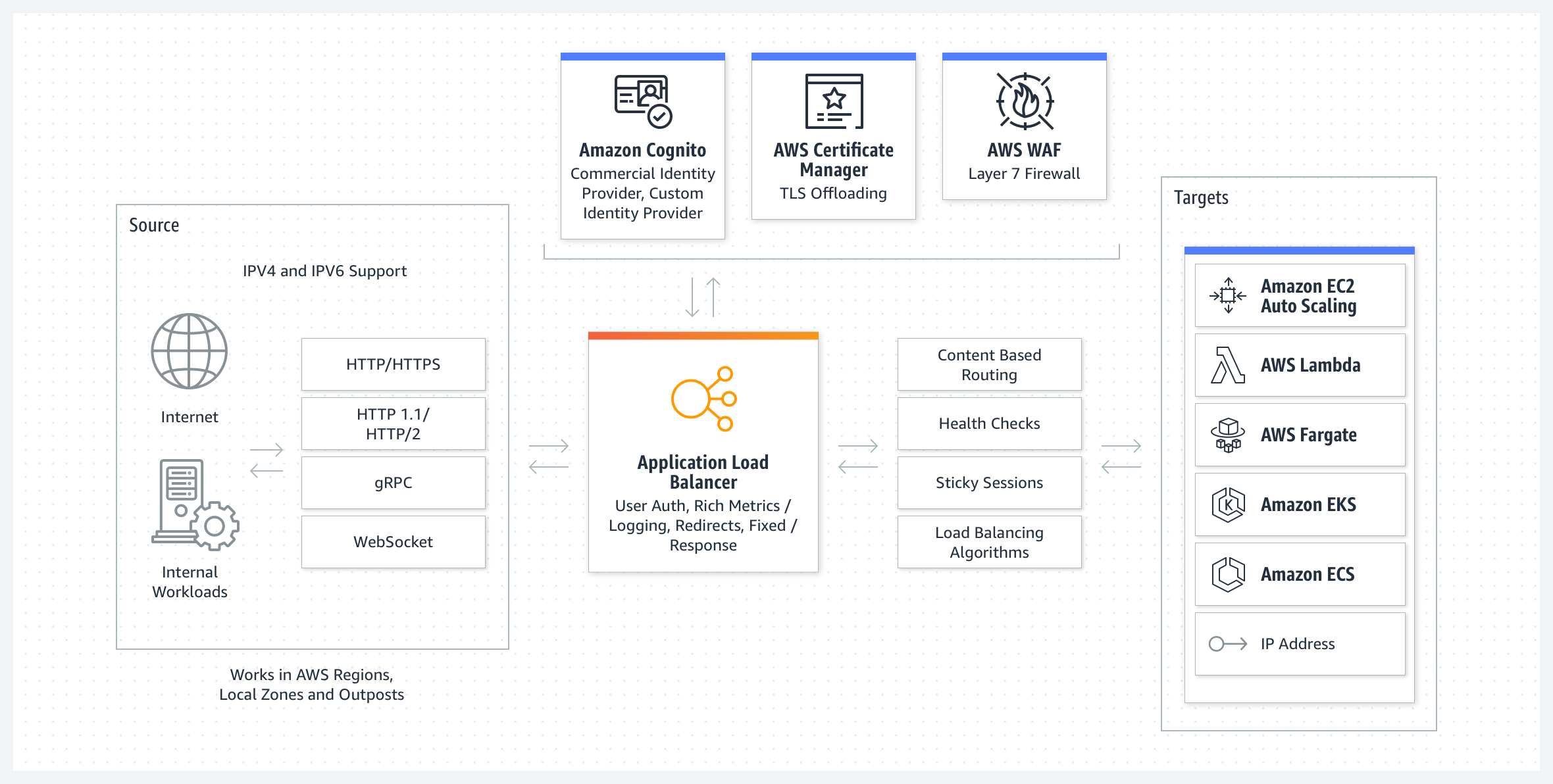
○ Automatically scale resources based on demand.

AWS Auto Scaling

# How scaling plans work

AWS Auto Scaling lets you use scaling plans to configure a set of instructions for scaling your resources. If you work with AWS CloudFormation or add tags to scalable resources, you can set up scaling plans for different sets of resources, per application. The AWS Auto Scaling console provides recommendations for scaling strategies customized to each resource. After you create your scaling plan, it combines dynamic scaling and predictive scalin

<https://aws.amazon.com/autoscaling/>



5. **DevOps Considerations:** Describe how you would implement:

○ **Infrastructure as Code (IaC):** Tools and practices to automate infrastructure provisioning and management.

Terraform

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

<https://developer.hashicorp.com/terraform/intro>

Amazone Infrastructure as Code

<https://aws.amazon.com/what-is/iac/>

○ **CI/CD Pipelines:** Processes for continuous integration and deployment to ensure smooth releases and rapid feedback loops.

GitHub Actions Workflow

Automate, customize, and execute your software development workflows right in your repository with GitHub Actions. You can discover, create, and share actions to perform any job you'd like, including CI/CD, and combine actions in a completely customized workflow

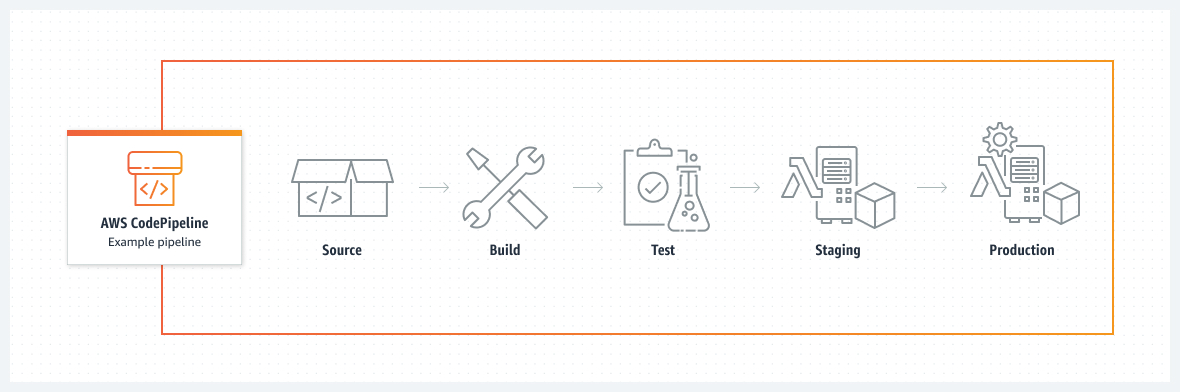
<https://docs.github.com/en/actions>

AWS CodePipeline

<https://aws.amazon.com/codepipeline/>

## How it works

AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates.



○ **Monitoring and Logging:** Mechanisms for collecting logs, monitoring system health, and alerting on issues.

# <https://docs.aws.amazon.com/prescriptive-guidance/latest/implementing-logging-monitoring-cloudwatch/welcome.html>

# Designing and implementing logging and monitoring with Amazon CloudWatch

This guide helps you design and implement logging and monitoring with [Amazon CloudWatch](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html) and related Amazon Web Services (AWS) management and governance services for workloads that use [Amazon Elastic Compute Cloud (Amazon EC2) instances](https://docs.aws.amazon.com/ec2/index.html), [Amazon Elastic Container Service (Amazon ECS)](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html), [Amazon Elastic Kubernetes Service (Amazon EKS)](https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html), [AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html), and on-premises servers. The guide is intended for operations teams, DevOps engineers, and application engineers that manage workloads on the AWS Cloud.