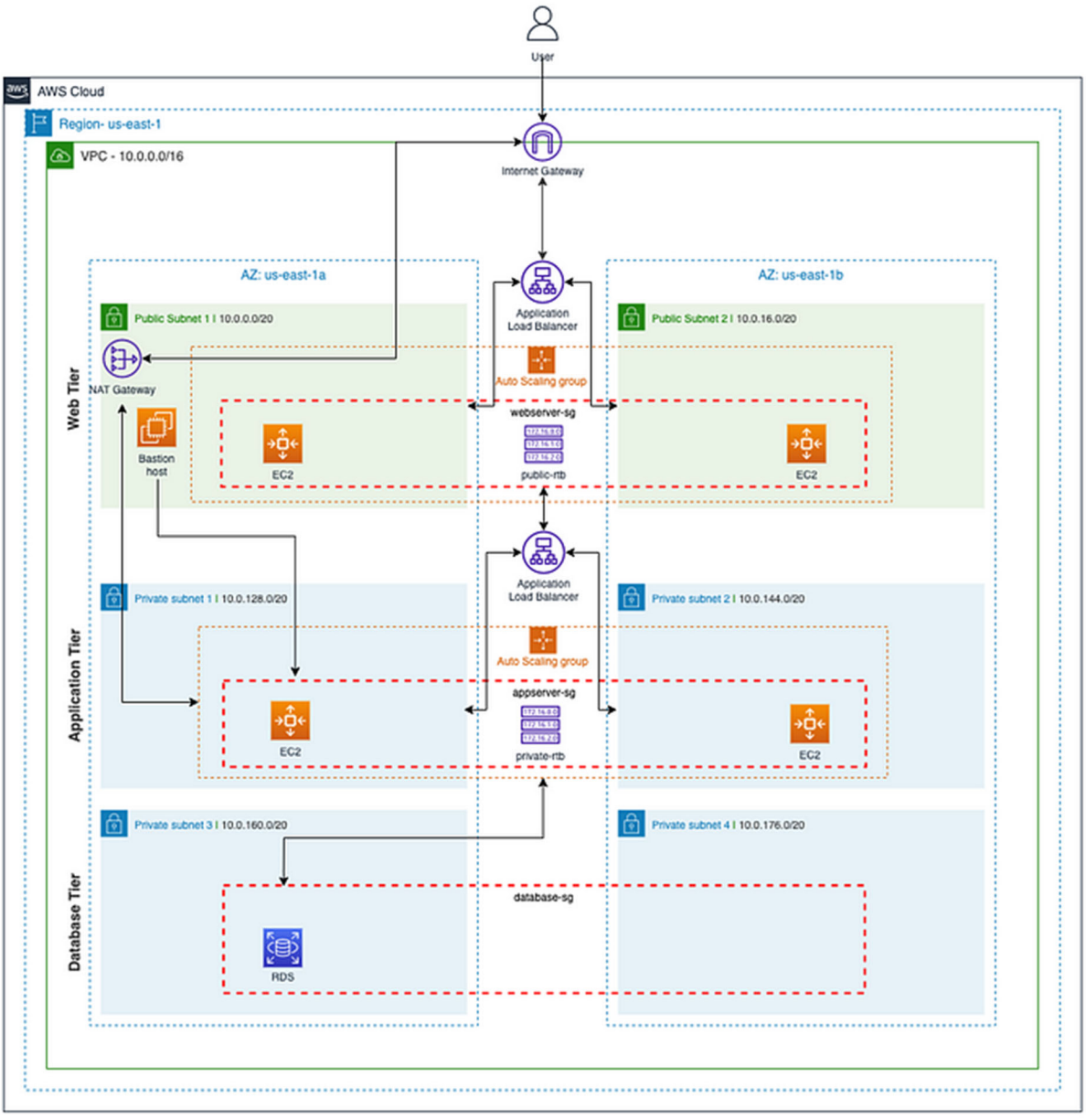
Building a 3-tier web application architecture with AWS

Diagram for Java Spring based :-



Why I have chosen AWS native service:-

I have chosen AWS native services over lift and shift can lead to a more efficient, cost-effective, and scalable 3-tier architecture on AWS. Native services are designed to leverage the cloud's capabilities and can help me take full advantage of what AWS has to offer. AWS native services are purpose-built for the cloud, designed to take full advantage of the AWS infrastructure. They are optimized for scalability, high availability, and performance, which may not be achievable with a lift-and-shift approach. Native AWS services offer pricing models that can be more cost-effective than running equivalent on-premises or virtualized services. AWS native services can automatically scale up or down based on demand. This elasticity is challenging to achieve with traditional on-premises applications and can help optimize resource utilization and cost. AWS provides built-in high availability features, such as multi-Availability Zone (AZ) deployments and automated failover. These features can enhance the reliability of our application without requiring additional infrastructure.

A three-tier architecture is well-suited for deployment on Amazon Web Services (AWS) due to its alignment with several AWS services and best practices.

Here is why the three-tier architecture is a good fit for AWS:

1. **Virtual Private Cloud (VPC):** AWS VPC enables to create isolated network environments, providing strong network-level security and segmentation between the tiers.

2. **Identity and Access Management (IAM):** IAM provides fine-grained access control for AWS resources, ensuring that only authorized entities can interact with our resources. Security Groups and Network ACLs: AWS offers security groups and network access control lists (NACLs) to control inbound and outbound traffic at the network level, enhancing security.

3. **CloudWatch and AWS CloudTrail**:

AWS provides services like Amazon CloudWatch and AWS CloudTrail for monitoring and auditing our infrastructure and applications. You can gain insights into application performance, troubleshoot issues, and ensure compliance.

4.Auto-scaling:

AWS provides **auto-scaling** capabilities that align perfectly with the three-tier architecture's scalability requirements. Each tier can be scaled independently based on the specific needs of the application.

5.ELB : AWS Elastic Load Balancing (ELB) allows for efficient distribution of incoming traffic across multiple instances in the application and presentation tiers, ensuring availability and scalability.

6.Multi-Availability Zone Deployment: AWS offers multiple Availability Zones within regions, enabling you to deploy each tier in separate zones for redundancy. In case of failure in one zone, traffic can be automatically redirected to healthy instances in another zone.

7.Data Replication: AWS provides database replication options (e.g., Multi-AZ deployments for Amazon RDS) to ensure high availability and data durability in the data tier.

3-tier spreads into multiple Availability Zones and separating it into three layers that serve different functions, independent of each other. If an AZ does down for some reason, the application could automatically scale resources to another AZ, without affecting the rest of the application tiers. Each tier has its own security group that only allows the inbound/outbound traffic needed to perform specific tasks.

8.Web/Presentation Tier: User-facing elements of the application, such as web servers and the interface/frontend.

a.) Application Tier: Backend and application source code needed to process data and run functions.

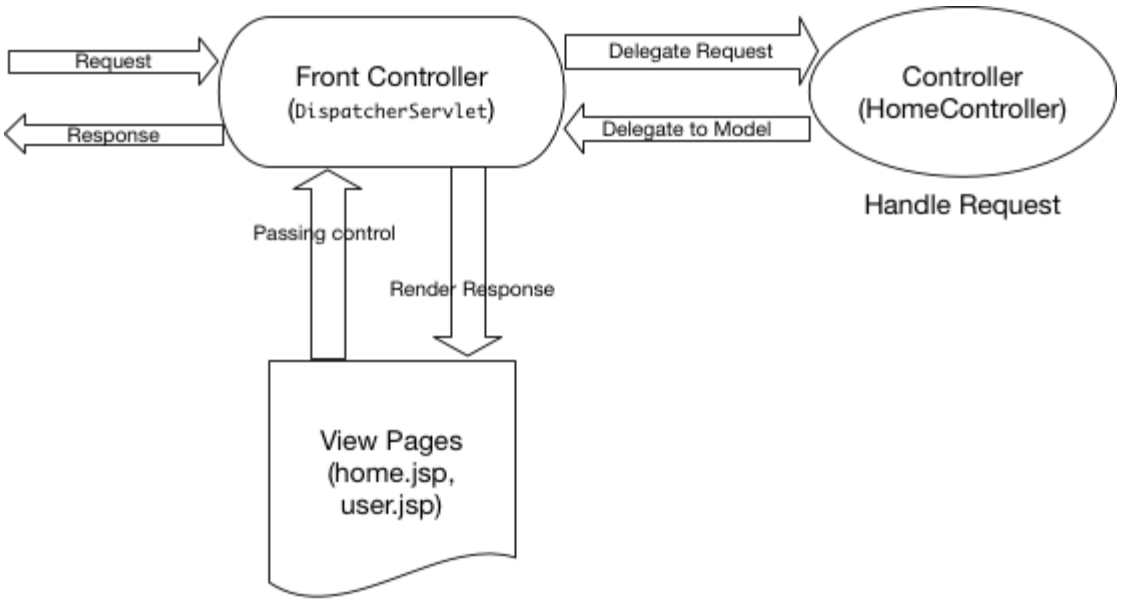
b.) Database Tier: Manages the application data. Often where the databases are stored.

End to End communication between three tiers: In a Spring MVC web application, the presentation tier (also known as the web tier) communicates with the Spring Java application in the backend (also known as the service or application tier) using HTTP requests and responses. The communication is typically handled by Spring MVC controllers in the presentation tier, which receive HTTP requests from clients (e.g., browsers) and interact with the backend services to process those requests and generate responses.

Here is a step-by-step example of how the presentation tier communicates with the backend in a Spring MVC application:

Client Sends HTTP Request: A user interacts with the web application by clicking a link, submitting a form, or making an HTTP request through their web browser.

Spring MVC Controller in Presentation Tier: The incoming HTTP request is received by a Spring MVC controller in the presentation tier. Controllers are responsible for handling different types of requests and deciding how to process them.



Example Controller:

```
@Controller
public class MyController {
    @Autowired
    private MyService myService;
    @GetMapping("/example")
    public String handleRequest(Model model) {
        // Controller logic here
        String result = myService.processRequest();
        model.addAttribute("result", result);
        return "viewName"; // Returns the name of the view to render
    }
}
```

Controller Invokes Backend Service:The controller typically delegates the processing of the request to a backend service. In the example above, the `MyService` component is injected into the controller.

Example Service:

```
@Service
public class MyService {
    public String processRequest() {
        // Service logic here
        return "Processed Data";
    }
}
```

Backend Service Performs Business Logic: The backend service performs business logic, interacts with databases, or other services as needed to fulfill the request. In this example, it returns "Processed Data."

Service Returns Response to Controller: The backend service returns the result to the controller in the presentation tier.

Controller Prepares Response: The controller processes the result from the backend service, prepares a response (e.g., a model with data), and selects a view to render.

View Rendering: The selected view (e.g., a JSP, Thyme leaf template, or HTML file) is rendered, incorporating the data from the model.

HTML Response Sent to Client: The HTML response is sent back to the client's web browser, which displays the content.

Why I chosen EC2 for Web tier and App tier:-

EC2 is a good choice for our legacy applications. These legacy applications may not be designed to run in containerized environments or serverless architectures. EC2 instances provide a traditional computing environment that can support these legacy applications without major modifications. EC2 instances give us full control over the underlying virtual machines, including the choice of operating system, software configurations, and network settings. This level of control is often necessary for running older or specialized software. Legacy applications may have specific dependencies, configurations, or software stacks that are challenging to replicate in a containerized or serverless environment.

With EC2, we can tailor the environment to match the requirements of the legacy software. Migrating legacy databases and data stores to the cloud is often a complex task. EC2 instances allow we have to retain control over our database servers and storage solutions, making data migration more manageable. EC2 instances can serve as an intermediate step during a larger migration or modernization project. we can gradually refactor or re-architect our legacy application while still maintaining its functionality on EC2 instances. For legacy applications with limited development and maintenance resources, running on EC2 instances may require minimal code changes compared to containerization or serverless migration.

Legacy applications may rely on specific network configurations or interfaces that are easier to maintain on EC2 instances. We can configure Virtual Private Cloud (VPC) settings and networking components to match the legacy application's requirements. Some legacy applications may have dependencies on specific hardware components or peripherals that are challenging to replicate in a cloud-native environment. EC2 instances can be configured with the necessary hardware specifications.

Why do I need to use NAT Gateway redundant :-

A NAT Gateway (Network Address Translation Gateway) is a managed AWS service that allows resources in a private subnet of a Virtual Private Cloud (VPC) to initiate outbound internet traffic while maintaining security by not allowing inbound traffic initiated from the internet to reach those resources directly. NAT Gateway acts as an intermediary between private resources and the public internet. Redundancy in network components like NAT Gateways is crucial for ensuring high availability. If a single NAT Gateway becomes unavailable due to a failure or maintenance, it can disrupt outbound internet access for resources in the associated private subnet.

Why do we need Bastion Host: -

A Bastion Host, also known as a jump host or jump server, is a specialized instance in an Amazon Web Services (AWS) Virtual Private Cloud (VPC) or any network environment. Its primary purpose is to provide a secure and controlled entry point for administrators to access and manage other resources, such as instances, services, or databases, located in private subnets within the VPC. Bastion Hosts enhance security by serving as a single point of entry for administrators to access resources in private subnets. This prevents direct external access to private instances and helps protect them from unauthorized access

Ingress and Egress of public Subnet via Public Router table

Public route tables are typically associated with subnets that contain resources (such as web servers, load balancers, or NAT gateways) that need to be directly accessible from the internet. To enable internet access, a public route table usually includes a default route (0.0.0.0/0) pointing to an internet gateway. This allows outbound traffic from the associated subnets to reach the internet and incoming internet traffic to reach the resources in the public subnets. Ingress (incoming) traffic from the internet is directed to resources in public subnets based on their Elastic IP addresses, public IP addresses, or through an Elastic Load Balancer (ELB) or Application Load Balancer (ALB). Egress (outgoing) traffic from resources in public subnets is routed through the internet gateway. Public-facing web applications, load balancers, and resources that need to download software updates from the internet often reside in public subnets with public route tables.

Ingress and Egress of private Subnet via Public Router table

Private route tables are associated with subnets that contain resources (such as application servers, databases, and backend services) that do not require direct internet access. To enable outbound internet access for resources in private subnets (e.g., for software updates), a private route table often includes a default route pointing to a Network Address Translation (NAT) gateway or NAT instance located in a public subnet. This allows resources in private subnets to reach the internet while keeping them hidden from incoming internet traffic. To enable outbound internet access for resources in private subnets (e.g., for software updates), a private route table often includes a default route pointing to a Network Address Translation (NAT) gateway or NAT instance located in a public subnet. This allows resources in private subnets to reach the internet while keeping them hidden from incoming internet traffic. Ingress traffic to resources in private subnets is typically controlled by security groups and network ACLs. Egress traffic flows through the NAT gateway or NAT instance in the public subnet before reaching the internet. Application servers, backend databases, and internal services that should not be directly exposed to the internet often reside in private subnets with private route tables.

Role of S3

In a 3-tier architecture on Amazon Web Services (AWS), Amazon S3 (Simple Storage Service) can be connected at various points within the architecture to store and manage static assets, data, and files. Here are common use cases for connecting S3 in a 3-tier architecture:

Front-End Tier (Web Tier) === Static Content Hosting: S3 is commonly used to host static web assets such as HTML, CSS, JavaScript, images,

videos, and downloadable files. You can serve these assets directly from S3, reducing the load on the web server and improving performance.

Application Tier (Middle Tier) == Data Storage: S3 can be used to store application data, backups, and data snapshots. This provides a durable and scalable storage solution for our application's data needs.

Back-End Tier (Database Tier)== Data Backups: You can use S3 to store database backups, which provides a cost-effective and durable backup solution for our database instances.

Communication Between Tiers==Data Transfer: S3 can serve as a data transfer point between the tiers. For example, the application tier can upload files to S3, and the front-end tier can download and display those files. S3 provides a reliable and scalable mechanism for transferring data between tiers.

CDN Integration==: You can integrate Amazon CloudFront (AWS's content delivery network) with S3 to cache and distribute static content globally, reducing latency and improving content delivery to end-users.

Why I have chosen ALB

Amazon Web Services (AWS) Application Load Balancer (ALB) is a powerful and flexible load balancing service designed for modern application architectures. Choosing an ALB over other load balancers in AWS can offer several advantages:

ALB operates at the application layer (Layer 7) of the OSI model. It can route traffic based on content, URL path, host, HTTP headers,

and cookies. This level of granularity is essential for modern web applications with complex routing requirements. ALB can terminate SSL/TLS encryption for incoming requests, offloading the SSL handshake process from our application servers and simplifying certificate management. ALB provides access logs and metrics that can be used for monitoring, debugging, and performance optimization. It integrates with AWS CloudWatch for centralized logging and monitoring. ALB provides access logs and metrics that can be used for monitoring, debugging, and performance optimization. It integrates with AWS CloudWatch for centralized logging and monitoring. ALB can be configured to work with AWS Web Application Firewall (WAF) to protect against common web application security threats, such as SQL injection and cross-site scripting (XSS) attacks. ALB is designed for high availability and can automatically scale based on traffic demand. It distributes traffic evenly across healthy instances. ALB pricing is based on the number of Application Load Balancer Capacity Units (LCUs) used, making it a cost-effective choice for most workloads. While AWS Classic Load Balancer (CLB) and Network Load Balancer (NLB) have their use cases, ALB is often the preferred choice for modern web applications that require advanced routing, content-based routing, and application-layer features.

While AWS Classic Load Balancer (CLB) and Network Load Balancer (NLB) have their use cases, ALB is often the preferred choice for modern web applications that require advanced routing, content-based routing, and application-layer features. RDS supports multiple popular relational database engines, including Mysql. RDS supports multi-Availability Zone (multi-AZ) deployments to enhance availability and failover capabilities. RDS for MySQL includes automated backups, automated software patching, and read replicas.

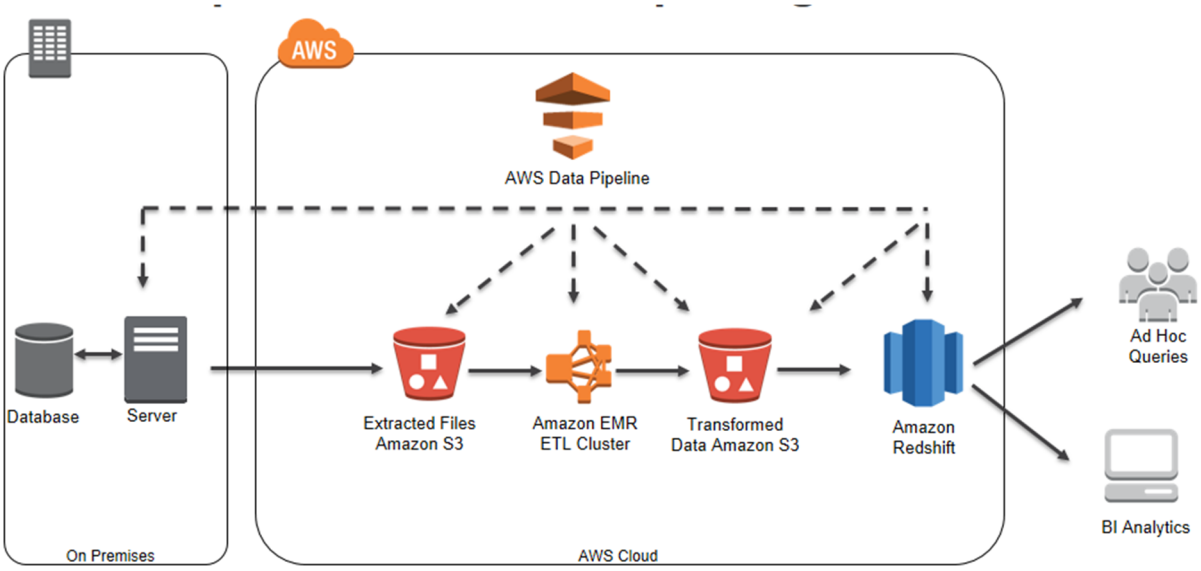
Security groups

Security groups are required for resources in both public and private subnets within an Amazon Web Services (AWS) Virtual Private Cloud (VPC). Security groups are an essential part of AWS's network security model, and they help control inbound and outbound traffic to and from our resources. Even though resources in a private subnet are not directly accessible from the internet, they can communicate with each other and with resources in public subnets. Security groups allow you to define rules that control which resources are allowed to initiate inbound connections to resources in the private subnet. This ensures that only authorized traffic is permitted. Security groups also control outbound traffic from resources in the private subnet. We can define rules that specify the destinations and ports to which resources in the private subnet are allowed to connect. This helps prevent unauthorized outbound traffic. Security groups provide visibility into network traffic by allowing you to log rejected traffic. This can be helpful for security auditing and monitoring.

Amazon RDS (Relational Database Service) for MySQL

Creating a redundant and highly available Amazon RDS (Relational Database Service) for MySQL in AWS involves setting up a Multi-Availability Zone (Multi-AZ) deployment. In a Multi-AZ configuration, our primary RDS instance is automatically replicated to a standby instance in a different Availability Zone (AZ) within the same AWS region. This provides data redundancy, failover capabilities, and increased availability. Here is how to create a redundant RDS for MySQL. This setup provides redundancy, high availability, and data durability for our MySQL database in AWS.

Data Analytics Workload



To spin up an Amazon EMR (Elastic MapReduce) cluster for a data analytics solution that currently runs on Hadoop, we can follow these steps to design an architecture that aligns with best practices and efficiently utilizes AWS services:

Assess Requirements:

Identify the specific Hadoop components and configurations used in our existing Hadoop cluster. Determine the data sources, data formats, and storage solutions involved in our analytics workload. Understand the types of analytics jobs we need to run, such as batch processing or real-time processing.

Amazon EMR:

We need to create an EMR cluster with the appropriate instance types, cluster size, and version of Hadoop/Spark that matches our workload requirements

This managed Hadoop and Spark service is the core of our architecture. It provides scalability and simplifies the deployment and management of Hadoop clusters.

Configure our EMR cluster to use our Amazon S3 data lake as the data source

Data Storage:

Amazon S3: Use Amazon S3 as our primary data lake to store data efficiently and cost-effectively. It integrates seamlessly with EMR.

Amazon Redshift: data warehousing services for structured data storage and processing, if needed.

Implement backup and recovery strategies for our data and analytics workloads using Amazon S3 versioning and cross-region replication

Data Ingestion and ETL:

Use AWS Glue to ingest and transform data from various sources into the data lake (Amazon S3). Schedule ETL jobs using AWS Glue.

Analytics Jobs: Run our Hadoop-based analytics jobs (e.g., MapReduce, Hive, Pig) on the EMR cluster. Utilize Spark for faster

data processing and machine learning tasks, as it is well-integrated with EMR.

AWS Glue and Amazon EMR are complementary services that work together to create end-to-end data processing pipelines. Glue handles data preparation and ETL, while EMR is used for more complex data processing and analytics tasks, making it a powerful combination for handling large-scale data workloads

Monitoring and Optimization:

Set up monitoring using Amazon CloudWatch to track cluster performance, resource utilization, and job progress.

Use AWS Auto Scaling to automatically adjust the cluster size based on demand.

Optimize our Hadoop and Spark configurations for performance.

Visualization and Reporting:

Connect our preferred visualization tools (e.g., Tableau, QuickSight) to EMR or use Amazon QuickSight to create dashboards and reports. Visualize insights derived from our analytics jobs.