Problem Statement Approach

Infrastructure Roadmap:

- We will setup the environment using **public cloud AWS**.
- First, we will setup our network in AWS using **Virtual Private Cloud**, create subnets in multiple Availability zones for high availability.
- We will take the **Ubuntu** machines as our servers, which will be available **24*7**.
- All servers will be private in nature so that traffic can come from only **Load Balancer**.
- For some configuration in future, we will setup a **Bastion host** in a public subnet.
- For static content, the requests will be handled by **Nginx**.
- Nginx will have **HTTPS** enabled through **self signed certificates**.
- For dynamic content, the requests will be handled by **Apache Tomcat**.
- **Security Group** attached to our servers will have only **port 443** for HTTPS and **8080** for Apache Tomcat and all other ports will be by default disabled.
- And the traffic can come from only Load Balancer.
- We will create a **Golden AMI** in AWS which will have all the required packages for our web application.
- For scaling our servers, we will configure Autoscaling by creating an **AWS AutoScaling Group** connected to our **launch template/configuration**(which will have our Golden AMI, instance type, SSH-key pair, subnets and security group).
- We will configure Scaling policies based on **Average CPU utilization**.
- A **Load Balancer** will be setup on our instances which will be grouped together through **AWS Target Group**.
- An **Application Load Balancer** will be setup which will handle all requests through path based routing.
- A **Network Load Balancer** will be setup above the Application Load Balancer so that a large no. of requests can be handled smoothly.
- We will regiter a **domain** for our java web application whose routing policies will be handled by **Amazon Route 53**.
- We will also setup Content Delivery Network using **Amazon CloudFront** on our Network Load Balancer for ultra low latency to our customers.

Infrastructure Provisioning:

- We will manage the infrastructure provisioning through **Terraform**, creating **seperate modules** for each group of resources.
- We will push our terraform code to a **Version Control Service** offered by **GitHub**.
- We will manage the **terraform backend/state file** in a remote location which will be **AWS S3 Bucket**, so that multiple Engineers can work on one piece of Code together.
- We will take an already created Bucket(private in nature) which has **encryption at rest** and **bucket versiong enabled**.
- We will also setup **Bucket replication** policy so that our terraform state file has a **backup** in another bucket which will be in a **different region** from the original bucket.
- We will lock the terraform state file using **AWS DynamoDB** so that at a point of time, only one developer can execute the code.
- We will take an already created **DynamoDB** whose name we will pass in terraform backend block in our terraform code.

CI/CD pipeline

➓ We will apply the code through automation using **CI/CD pipeline** which we will achieve using **GitHub Actions**.

➓ Terraform code's validation and plan will be run automatically. If the plan runs successfully, we will **apply the code** manually using "**workflow_dispatch**" event in GitHub actions.