Question:
If you were to scale your customer API service to millions of customers how would you build it differently?

Answer:

We will leverage Kubernetes for orchestration, scaling, and management to scale this application to handle millions of customers. This approach ensures scalability, load balancing, real-time monitoring, and a streamlined deployment process. Below is a concise breakdown:

Containerization (Docker)

- Dockerize the application with a Dockerfile.
- Build and push the Docker image to a container registry (e.g., DockerHub, GitLab Container/image Registry).

Kubernetes Deployment
- Deploy the application using a Kubernetes Deployment manifest.
- Use a ClusterIP Service to expose the API within the cluster.

Ingress Controller
- Set up an NGINX Ingress Controller to handle external traffic.
- Define an Ingress Resource to manage load balancing and routing.

Auto-scaling
- Implement Horizontal Pod Autoscaler (HPA) to dynamically scale the number of pods based on CPU or memory usage.
- Set minimum and maximum replicas for pods based on traffic and resource consumption.

Monitoring
- Use Prometheus to collect metrics and Grafana for visualization.
- Install the Kubernetes Metrics Server to monitor real-time resource usage.

Logging (EFK Stack)
- Centralize logging using Elasticsearch, Fluentd, and Kibana (EFK) stack for better visibility and troubleshooting.

Persistent Storage
- Use Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) for MySQL to ensure data persistence.

CI/CD Pipeline

- Automate the deployment process with GitLab CI, TeamCity, Octopus, or GitHub Actions.
- Use SonarQube for code quality checks and code coverage analysis.