INSURANCE MANAGEMENT SYSTEM

The **Insurance Management System** is a microservices-based project that manages insurance policies and claims.

Users can apply for policies and raise claims, agents manage and verify them, while admins finalize approvals.

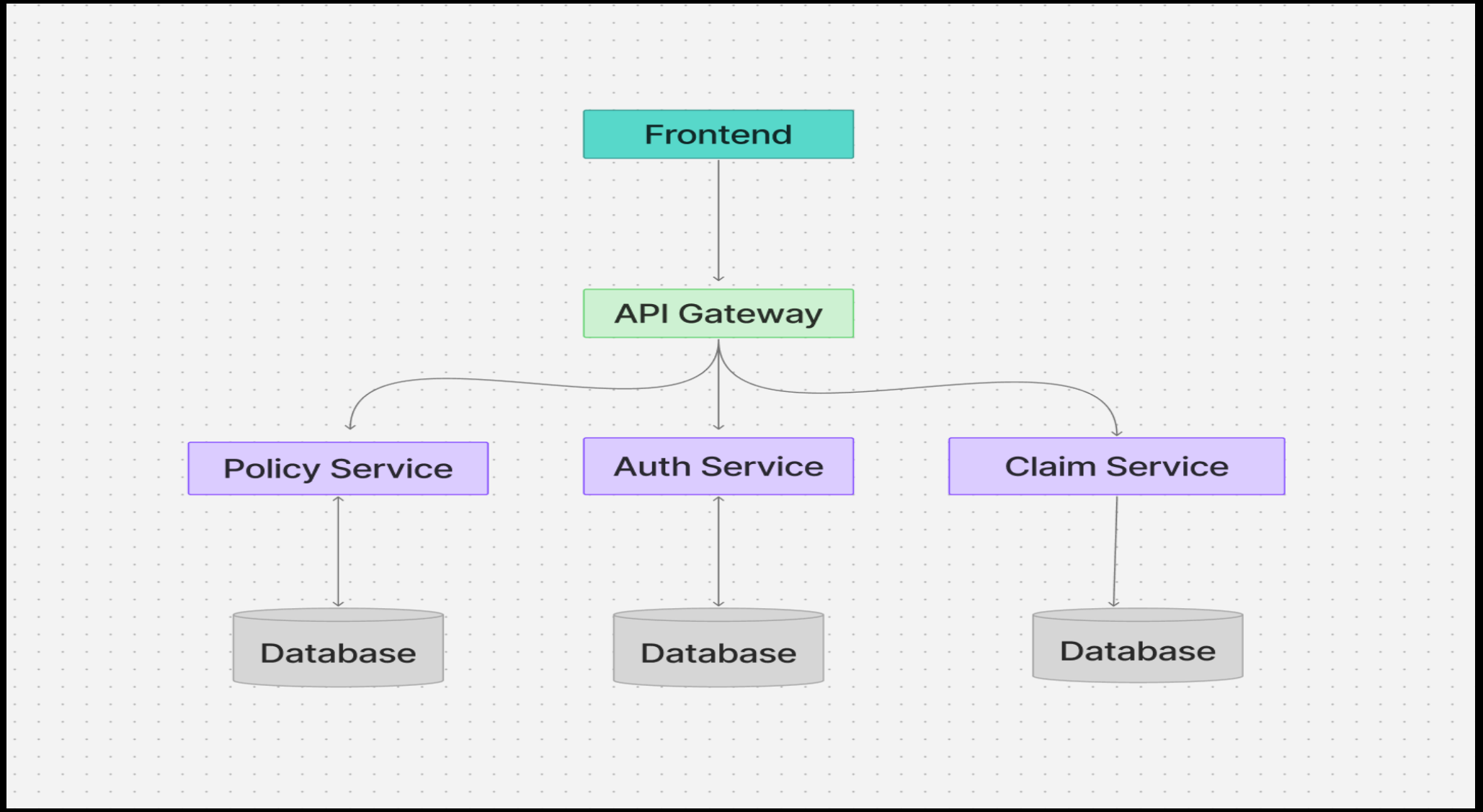
The system is containerized using **Docker**, deployed and managed with **Kubernetes**, and automated through **Jenkins CI/CD**.

ARCHITECTURE

MICROSERVICES

- The project follows a **Microservices Architecture**, where each functionality is developed as a **separate service**.
- Each service runs on its own **port** and has its own **database**, making them independent and scalable.
- An **API Gateway** is used as a single entry point to route requests to the right service.
- Services communicate using **REST APIs**.
- The system is packaged into **Docker containers** and deployed using **Kubernetes** for orchestration.
- **Jenkins CI/CD** automates building, testing, and deploying all services.

ARCHITECTURE



OVERVIEW EACH SERVICES AND PRESENT API'S EXPOSED

AUTH-SERVICE

Overview:

- Handles authentication and authorization.
- Manages **users, agents, and admins** with role-based access.

APIs:

- `POST /register` → Register a new user/agent.
- `POST /login` → Login and get authentication token.
- `GET /users` → List all registered users (admin only).
- `GET /users/{id}` → Get details of a user.

POLICY-SERVICE

- **Overview:**
- Manages **insurance policies**.
- Users can apply for policies.
- Agents review applications.
- Admins finalize approvals.
- **APIs:**
- `GET /policies` → Get all policies.
- `GET /policies/{id}` → Get details of one policy.
- `POST /policies` → Apply for a new policy.
- `PUT /policies/{id}` → Update policy (agent/admin).
- `DELETE /policies/{id}` → Cancel a policy.

CLAIM-SERVICE

OVERVIEW:

- HANDLES **INSURANCE CLAIMS**.
- USERS RAISE CLAIMS FOR APPROVED POLICIES.
- AGENTS VERIFY CLAIMS.
- ADMINS TAKE THE FINAL DECISION.

APIS:

- `GET /LIST-CLAIMS` → GET ALL CLAIMS.
- `POST /CREATE-CLAIMS` → RAISE A NEW CLAIM.
- `PUT /CLAIMS/{ID}` → UPDATE CLAIM STATUS (AGENT/ADMIN).
- `DELETE /CLAIMS/{ID}` → REMOVE A CLAIM.

API GATEWAY

Overview:

- Acts as the **single entry point** for all clients (User, Agent, Admin).
- Forwards requests to the right microservice (`auth-service`, `policy-service`, `claim-service`).
- Provides **centralized authentication, routing, and security**.
- Makes the system **easier to use and more secure**.

APIs (routes exposed):

- `POST /auth/register` → Forward to **Auth-Service** (register a user/agent/admin).
- `POST /auth/login` → Forward to **Auth-Service** (login).
- `GET /policies` → Forward to **Policy-Service** (list all policies).
- `POST /policies` → Forward to **Policy-Service** (apply for new policy).
- `GET /claims` → Forward to **Claim-Service** (list all claims).
- `POST /claims` → Forward to **Claim-Service** (raise a new claim).

CONTAINERIZATION – DOCKER

- Each service is packaged into a **Docker container**.
- Containers make the system **lightweight, portable, and consistent** across environments.
- Every service runs in its **own container** with its dependencies.
- Containers communicate with each other through **Docker networking**.

Containers Created:

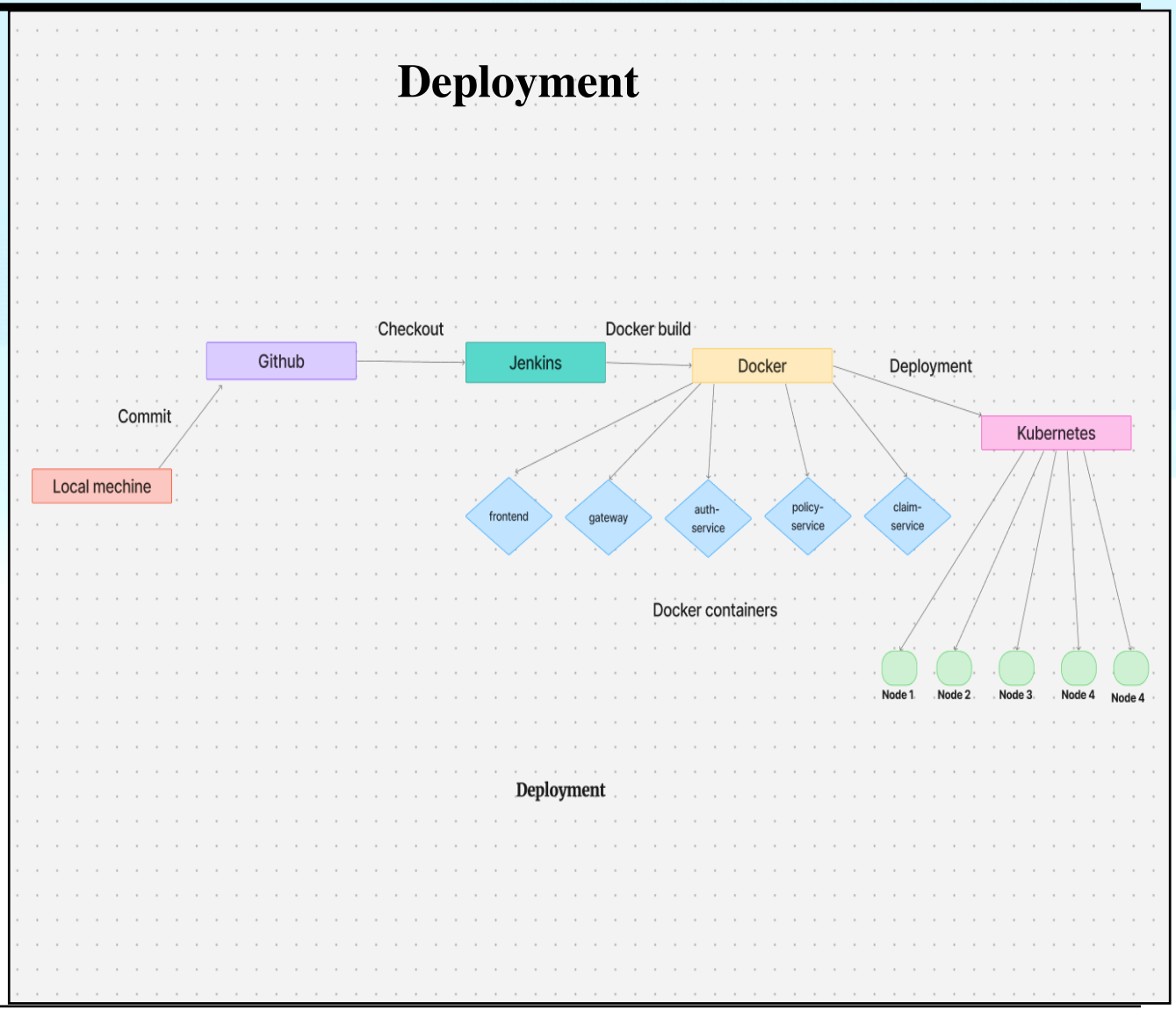
- **API Gateway Container** → Handles routing and authentication.
- **Auth-Service Container** → Manages login, registration, and role-based access.
- **Policy-Service Container** → Handles insurance policy applications.
- **Claim-Service Container** → Manages insurance claims.
- **Frontend Container** → Provides UI for Users, Agents, and Admins.

JENKINS IN THE ARCHITECTURE :

- It automates the **CI/CD pipeline**:
- Watches the GitHub repo for code changes.
- Builds Docker images for each microservice.
- Runs tests to ensure services work correctly.
- Deploys the containers to Docker Compose or Kubernetes.

DEPLOYMENT

- The system is deployed using **Docker Compose** (for local setup) and **Kubernetes** (for orchestration in production).
- **Jenkins CI/CD pipeline** automates deployment by building Docker images, running tests, and deploying containers.
- Each microservice runs in its own **container** with its dependencies.
- Kubernetes manages **scaling, load balancing, and service discovery**.



THANK YOU

Sourav Roy

ID - 30437

roys45545@gmail.com