**MediLink — Intelligent Healthcare Data Exchange & Workflow Orchestration Platform**

**Executive summary (one-line for CV)**

Developed **MediLink**, a secure, event-driven healthcare data exchange and workflow orchestration platform using Spring Boot microservices, Kafka, RabbitMQ, PostgreSQL, Redis, Docker, Kubernetes, and enterprise patterns (CQRS, event sourcing, sagas, OpenTelemetry, FHIR interoperability, and HIPAA-grade security).

**1. Formal Project Proposal**

**1.1 Introduction**

MediLink is a microservice-based platform that connects hospitals, labs, pharmacies, and insurers to enable real-time, auditable healthcare workflows. It uses an event-driven backbone (Kafka) and task queues (RabbitMQ) to orchestrate multi-actor processes like lab orders, prescriptions, and insurance claims. Focus areas: scalability, resilience, security, observability, and FHIR interoperability.

**1.2 Problem Statement**

Healthcare organizations operate in silos with incompatible systems causing delays and errors in time-sensitive workflows (lab results, prescriptions, claims). Batch ETL or point-to-point integrations are brittle and slow. MediLink provides a standardized, real-time exchange and workflow orchestration layer that ensures reliability, observability, compliance, and interoperability.

**1.3 Project Objectives**

1. Provide real-time event-driven routing for clinical workflows (admissions, lab orders/results, prescriptions, claims).
2. Use Kafka as the durable event bus and RabbitMQ for background tasks such as notification and report generation.
3. Implement CQRS + Event Sourcing for critical business services to maintain an immutable audit trail.
4. Orchestrate multi-step distributed transactions using sagas (choreography or orchestration).
5. Ensure robust security: OAuth2/JWT, RBAC, mTLS between services, encrypted secrets.
6. Full observability: metrics, logs, and distributed tracing (OpenTelemetry, Prometheus, Grafana, Jaeger).
7. FHIR-based APIs for data exchange.

**1.4 Scope (MVP)**

* Core microservices (Patient, Lab, Prescription, Insurance, Notification, Audit, Auth).
* Local dev using Docker Compose (Kafka + Zookeeper, RabbitMQ, Postgres, Redis).
* Kubernetes deployment with Helm charts.
* Basic Angular admin UIs (Doctor, Lab, Insurance, Pharmacy).
* CI/CD with GitHub Actions and ArgoCD for GitOps.
* Observability stack for metrics, traces, and logs.

**1.5 Methodology / Architecture Overview**

* **Event bus**: Kafka topics per domain (patient.events, lab.orders, prescriptions, claims).
* **Task queue**: RabbitMQ for async background work (PDF generation, external notifications).
* **Microservices**: Each bounded context is a Spring Boot service using a local Postgres DB; commands write events; events go to Kafka; read models maintained by consumers.
* **Sagas**: SAGA coordinator service (optional) for orchestrated workflows or use choreography with stateful sagas stored in DB/Redis.
* **Auth**: Keycloak or Spring Authorization Server for OAuth2/OIDC and RBAC.
* **Caching**: Redis for read-model caching & distributed locks (idempotency keys).
* **Observability**: OpenTelemetry instrumentation in Java and Angular apps, Prometheus + Grafana, Jaeger traces, Loki/ELK for logs.
* **Secrets**: HashiCorp Vault or Kubernetes Secrets with sealed-secrets for production.

**1.6 Tech Stack**

* Backend: Java, **Spring Boot**, Spring Cloud (Kafka integration), Resilience4j.
* Messaging: **Apache Kafka**, **RabbitMQ**
* Database: **PostgreSQL** per service.
* Cache: **Redis**.
* Frontend: **Angular** (admin dashboards).
* Containerization: **Docker**, local dev docker-compose.
* Orchestration: **Kubernetes** + **Helm**.
* CI/CD: **GitHub Actions**, **Docker Registry** (DockerHub/GCR/ACR), **ArgoCD** (GitOps deploy).
* Observability: **OpenTelemetry**, **Prometheus**, **Grafana**, **Jaeger**, **Loki**.
* Security: **Keycloak** / Spring Authorization Server, mTLS, Vault.

**1.7 Deliverables**

1. Microservice codebase skeleton (Spring Boot) for each domain.
2. Angular admin UI skeleton.
3. docker-compose.yml for local development.
4. Dockerfiles for services and frontend.
5. Helm charts for K8s deployment (dev/staging/prod).
6. GitHub Actions workflows for CI (build/test), CD (image push), and GitOps (ArgoCD trigger).
7. Architecture docs: ARCHITECTURE.md, RUNBOOK.md, and OpenAPI + FHIR schemas.
8. Observability dashboards and alert rules.
9. Security hardening checklist and sample network policies.

**1.8 Timeline (suggested, 12 weeks MVP)**

* Week 1: Architecture, design, repo scaffolding, dev environment (docker-compose).
* Week 2–3: Implement Patient Service (CQRS event sourcing MVP) + Postgres + Redis.
* Week 4: Lab & Prescription services + Kafka topics and producers/consumers.
* Week 5: Notification service (RabbitMQ consumers), Audit service (event store ingestion).
* Week 6: Auth (Keycloak) + RBAC + mTLS proof of concept.
* Week 7: Angular dashboards (basic flows).
* Week 8: Observability instrumentation (OpenTelemetry, Prometheus, Grafana, Jaeger).
* Week 9: K8s Helm charts, manifests, and local Kubernetes (kind/minikube) testing.
* Week 10: CI/CD pipelines + Container registry + ArgoCD setup.
* Week 11: Security hardening & compliance docs (audit trails, encryption).
* Week 12: QA, demo, docs, handover.

**1.9 Risks & Mitigations**

* **Data privacy/compliance risk** — mitigate with encryption-at-rest/in-transit, audit trails, RBAC, and secure secrets.
* **Complexity of distributed transactions** — adopt sagas + idempotency and use retries / circuit breakers.
* **Operational complexity (Kafka + K8s)** — use managed Kafka in prod or Kafka operator; provide robust runbooks and alerting.
* **Latency & duplication** — design idempotent consumers and deduplication strategies (unique keys + DB constraints).

**1.10 Conclusion**

MediLink addresses a meaningful domain problem and demonstrates deep enterprise skills: event-driven design, microservices, K8s deployment, observability, and strict security — great for a CV and enterprise capstone.

**2. Repo skeleton (suggested top-level layout)**

medilink/

├─ docs/

│ ├─ ARCHITECTURE.md

│ ├─ RUNBOOK.md

│ └─ FHIR\_SCHEMAS/

├─ infra/

│ ├─ docker-compose.yml

│ ├─ k8s/

│ │ ├─ charts/

│ │ │ ├─ patient-service/

│ │ │ └─ lab-service/

│ │ └─ base-manifests/

│ └─ argo/

├─ services/

│ ├─ patient-service/ (spring boot)

│ ├─ lab-service/

│ ├─ prescription-service/

│ ├─ insurance-service/

│ ├─ notification-service/

│ └─ audit-service/

├─ web/

│ └─ admin-angular/

├─ .github/

│ └─ workflows/

│ ├─ ci.yml

│ └─ cd.yml

└─ README.md

**3. Local dev: docker-compose.yml (example)**

Use this for local dev/testing. Replace images/tags for production.

version: '3.8'

services:

zookeeper:

image: confluentinc/cp-zookeeper:7.4.1

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

ports: ["2181:2181"]

kafka:

image: confluentinc/cp-kafka:7.4.1

depends\_on: [zookeeper]

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

ports: ["9092:9092"]

rabbitmq:

image: rabbitmq:3.10-management

ports: ["5672:5672","15672:15672"]

postgres:

image: postgres:15

environment:

POSTGRES\_USER: medilink

POSTGRES\_PASSWORD: medilinkpass

POSTGRES\_DB: medilink

ports: ["5432:5432"]

volumes:

- postgres-data:/var/lib/postgresql/data

redis:

image: redis:7

ports: ["6379:6379"]

keycloak:

image: quay.io/keycloak/keycloak:21.1.1

command: start-dev

environment:

KEYCLOAK\_ADMIN: admin

KEYCLOAK\_ADMIN\_PASSWORD: admin

ports: ["8080:8080"]

patient-service:

build: ./services/patient-service

environment:

SPRING\_DATASOURCE\_URL: jdbc:postgresql://postgres:5432/medilink

KAFKA\_BOOTSTRAP\_SERVERS: kafka:9092

RABBITMQ\_URI: amqp://rabbitmq:5672

REDIS\_HOST: redis

depends\_on: [postgres, kafka, rabbitmq, redis, keycloak]

ports: ["8081:8080"]

lab-service:

build: ./services/lab-service

environment:

SPRING\_DATASOURCE\_URL: jdbc:postgresql://postgres:5432/medilink

KAFKA\_BOOTSTRAP\_SERVERS: kafka:9092

REDIS\_HOST: redis

depends\_on: [patient-service]

ports: ["8082:8080"]

volumes:

postgres-data:

Note: Confluent images above include Zookeeper. For production prefer KRaft or managed Kafka.

**4. Example Dockerfiles**

**4.1 Spring Boot (service) Dockerfile**

FROM eclipse-temurin:17-jdk-jammy AS build

WORKDIR /app

COPY mvnw .

COPY .mvn .mvn

COPY pom.xml .

RUN ./mvnw -q -ntp dependency:go-offline

COPY src src

RUN ./mvnw -DskipTests package -Pprod -q

FROM eclipse-temurin:17-jre-jammy

ARG JAR\_FILE=target/\*.jar

COPY --from=build /app/${JAR\_FILE} /app/app.jar

EXPOSE 8080

ENTRYPOINT ["java","-jar","/app/app.jar"]

**4.2 Angular frontend Dockerfile**

# build stage

FROM node:18 AS build

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm ci

COPY . .

RUN npm run build -- --configuration production

# serve stage

FROM nginx:stable-alpine

COPY --from=build /app/dist/admin-angular /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

**5. Kubernetes & Helm (high-level)**

**5.1 Helm chart structure (example)**

infra/k8s/charts/

patient-service/

Chart.yaml

values.yaml

templates/

deployment.yaml

service.yaml

hpa.yaml

ingress.yaml

configmap.yaml

secret.yaml

pdb.yaml

**5.2 Example deployment.yaml (snippet)**

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ include "patient.fullname" . }}

spec:

replicas: {{ .Values.replicaCount }}

selector:

matchLabels:

app: {{ include "patient.name" . }}

template:

metadata:

labels:

app: {{ include "patient.name" . }}

version: {{ .Chart.AppVersion }}

spec:

containers:

- name: patient

image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

ports: [{ containerPort: 8080 }]

envFrom:

- secretRef: { name: patient-secrets }

- configMapRef: { name: patient-config }

livenessProbe:

httpGet: { path: /actuator/health/liveness, port: 8080 }

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet: { path: /actuator/health/readiness, port: 8080 }

initialDelaySeconds: 10

periodSeconds: 10

resources: {{ toYaml .Values.resources | nindent 12 }}

**5.3 Security & Networking**

* Use **NetworkPolicies** to restrict pod-to-pod traffic.
* Enable **mTLS** (Istio/Linkerd) or use service mesh for secure service-to-service comms.
* Configure **PodSecurityPolicies** / admission controllers and RBAC for least privilege.
* Use **Ingress** with TLS (certificate via cert-manager).

**6. CI/CD: GitHub Actions + ArgoCD (examples)**

**6.1 CI pipeline (.github/workflows/ci.yml)**

Build, test, and push images to registry.

name: CI

on:

push:

paths:

- 'services/\*\*'

- 'web/\*\*'

pull\_request:

jobs:

build:

runs-on: ubuntu-latest

env:

REGISTRY: ghcr.io/${{ github.repository\_owner }}

steps:

- uses: actions/checkout@v4

- name: Set up JDK 17

uses: actions/setup-java@v4

with:

java-version: '17'

distribution: 'temurin'

- name: Build and Test patient-service

working-directory: services/patient-service

run: |

./mvnw -B -DskipTests=false test package

- name: Build Docker images

uses: docker/build-push-action@v5

with:

context: .

push: true

tags: |

${{ env.REGISTRY }}/medilink/patient-service:${{ github.sha }}

${{ env.REGISTRY }}/medilink/lab-service:${{ github.sha }}

Store registry credentials in GitHub secrets (e.g., REGISTRY\_USER, REGISTRY\_TOKEN).

**6.2 CD pipeline (.github/workflows/cd.yml) — update Helm values and push to repo which ArgoCD watches**

name: CD

on:

push:

branches: [ main ]

paths:

- 'infra/k8s/\*\*'

- '.github/workflows/cd.yml'

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Update image tags in helm values

run: |

yq eval -i ".patientService.image.tag = \"${{ github.sha }}\"" infra/k8s/charts/patient-service/values.yaml

git config user.name "github-actions"

git config user.email "actions@github.com"

git add infra/k8s/charts/patient-service/values.yaml

git commit -m "ci: bump patient-service image to ${{ github.sha }}" || echo "no change"

git push

**ArgoCD** watches infra/k8s/ (GitOps): when this repo is updated, ArgoCD syncs to K8s cluster.

**7. Observability & Alerts (practical quick list)**

* Instrument Spring Boot apps with **OpenTelemetry**: spans for inbound HTTP, Kafka publish/consume, DB calls.
* Export to **Jaeger** for traces.
* Expose metrics via /actuator/prometheus and scrape with **Prometheus**.
* Build dashboards in **Grafana**: request latency, consumer lag, error rates, DB connections.
* Configure alerts:
  + High Kafka consumer lag → P1
  + High error rate (5xx) → P1
  + High memory usage / OOM → P2
  + Security alerts for failed auth attempts → P1

**8. Security checklist (MVP → Prod)**

* Use OAuth2/OIDC (Keycloak) with fine-grained RBAC.
* JWT validation in services; short-lived tokens + refresh tokens.
* mTLS for service-to-service communication via Mesh.
* Encrypt secrets (Vault) and use K8s sealed-secrets for GitOps.
* NetworkPolicies to limit traffic.
* Audit Service: immutable event log (append-only) with write-once storage.
* Data encryption at rest (Postgres TDE or storage-level).
* Penetration testing & threat model.

**9. Example: Lab test workflow (detailed)**

1. Doctor (Angular) hits Patient Service to create LabOrder (HTTP POST).
2. Patient Service writes LabOrderCreated event to Kafka (and persists command result in its event store).
3. Lab Service consumes LabOrderCreated, reserves technician, updates its local DB, emits LabProcessingStarted.
4. When result is ready, Lab Service emits LabResultReady.
5. Insurance Service consumes LabResultReady, starts claims verification; it emits InsuranceClaimSubmitted.
6. A Saga coordinates the process; on final approval, a LabResultPublished event triggers Notification Service (RabbitMQ) to generate PDFs and send notifications.
7. Audit Service consumes all events, writes immutable records to audit store.

**10. Example CV bullets (refined)**

* Designed and implemented **MediLink**, an event-driven healthcare workflow platform (Spring Boot, Kafka, RabbitMQ, Postgres, Redis) with CQRS, event sourcing, and sagas for distributed transactions.
* Built secure, FHIR-compliant APIs and immutable audit trails for HIPAA/GDPR compliance; deployed via Docker, Helm & Kubernetes with GitOps (ArgoCD) and a GitHub Actions CI pipeline.
* Implemented comprehensive observability (OpenTelemetry, Jaeger, Prometheus, Grafana) and resilient patterns (idempotency, DLQs, Resilience4j).

**11. Next steps I can do now (pick one — I’ll implement straight away)**

(You already gave two options; I made them explicit and added a third)

1. **Full project proposal document** (university/enterprise capstone): I’ll create a multi-page formal PDF/Markdown with Introduction, Problem, Objectives, Methodology, Architecture diagrams, Deliverables, Timeline, Risks, Budget & Appendix (detailed).
2. **Detailed repo skeleton + working docker-compose**: create a repository scaffold with minimal runnable Spring Boot example services (patient-service + lab-service) wired to Kafka/RabbitMQ/Postgres/Redis, plus Dockerfiles and docker-compose.yml that actually runs locally.
3. **Full CI/CD + K8s helm skeleton**: create production-ready GitHub Actions workflows, Helm charts for services, and example ArgoCD Application manifests to demonstrate GitOps deploy flow.

Tell me which option you want me to do next (1, 2, or 3). If you want a mix, say “2 + 3” and I’ll prioritize the repo + CI/CD/Helm artifacts together.