**🛠️ Project Phases (We Will Follow This Order)**

**📦 Phase 1: Environment Setup**

* Install Docker & Docker Compose
* Pull critoma/amd64\_u24\_noble\_ism\_security image
* Create Docker network
* Set up shared volume for image + output
* Define folder structure on host machine

**🔌 Phase 2: Core Containers Setup**

* **C01**: Create front-end upload + REST backend (Javalin or .NET)
* **C02**: Create RabbitMQ container with default topic/queue setup
* **C03**: Deploy MDB or POJO client (Java app with JMS subscriber) that:
  + Subscribes to messages
  + Launches native process (OpenMPI/OMP) to encrypt image

**🧠 Phase 3: Parallel Processing**

* Configure OpenMPI across **C03 and C04**
* Implement BMP AES encryption (ECB/CTR) with OpenMP
* Share workload via OpenMPI ranks

**🗃️ Phase 4: Data Persistence**

* **C05**: MySQL container with table for BLOB BMP files
* Optional: MongoDB for SNMP stats (CPU, RAM from other containers)

**🌐 Phase 5: Web Access and Notification**

* Node.js API for:
  + BMP rendering/download
  + SNMP stats view
* Notify frontend when encryption done (via REST or WebSocket)
* Frontend redirect to download page

C:\webcloud\_project

│

├── C01\_backend\

│ ├── src\main\java\ App.java <- Javalin app with upload form

│ │ ├── resources

│ ├── public\index.html <- Static HTML upload page

│ ├── pom.xml

│ └── Dockerfile <- For building the Javalin backend container

│

├── C02\_rabbitmq\

│ └── (No files needed) <- Will use official RabbitMQ image

│

├── C03\_processor\

│ ├── src\main\java\ ConsumerApp.java

│ │ ├── resources

│ ├── encrypt\_mpi.c <- C source for parallel AES encryption

│ ├── pom.xml

│ └── Dockerfile <- For building the consumer + native code

│

├── C04\_worker\

│ └── (No files needed)

│

├── volumes\

│ └── shared\ <- Shared volume for /tmp between C03 & C04

│ ├── input.bmp <- BMP to be encrypted

│ └── output.bmp <- Result written here

docker run -dit ^

--name c01\_backend ^

--hostname c01\_backend ^

-p 8080:8080 ^

--network mynet ^

c01\_backend\_img

docker run -dit ^

--name c02\_rabbitmq ^

--hostname c02\_rabbitmq ^

--network mynet ^

-e RABBITMQ\_DEFAULT\_USER=guest ^

-e RABBITMQ\_DEFAULT\_PASS=guest ^

-p 5672:5672 -p 15672:15672 ^

rabbitmq:3-management

docker run -dit --name c03\_processor --hostname c03\_processor --network mynet -v C:\webcloud\_project\volumes\shared:/tmp c03\_processor\_img

docker run -dit --name c04\_worker --hostname c04\_worker --network mynet -v C:\webcloud\_project\volumes\shared:/tmp critoma/amd64\_u24\_noble\_ism\_security

**📝 Original Task Breakdown (from your assignment)**

**🔢 Quoted Sections and ✅ Verification**

**🔸 "Create 5+ Docker containers from Docker Image... critoma/amd64\_u24\_noble\_ism\_security"**

✅ **Done**:

* c01\_backend
* c02\_rabbitmq
* c03\_processor
* c04\_worker
* volumes/shared used across C03+C04

✅ All are based on critoma/amd64\_u24\_noble\_ism\_security except RabbitMQ

**🔸 "Encrypting and Decrypting with symmetric AES key a large BMP picture"**

✅ **Done**:

* User uploads BMP + key/iv/mode
* BMP is saved to /tmp/input.bmp
* AES-ECB/CTR applied via encrypt\_mpi (OpenMP)
* Output saved to /tmp/output.bmp

**🔸 "Picture and symmetric key is received from the Front-end (Plain JS/CSS/HTML)... into the 1st container Back-End via REST API (Javalin)"**

✅ **Done**:

* c01\_backend handles form upload
* REST POST /upload accepts:
  + File
  + key
  + iv
  + mode
  + operation

**🔸 "...and then is published as binary message into a JMS/RabbitMQ Topic from the 2nd container..."**

✅ **Done**:

* c02\_rabbitmq runs
* Message sent to bmp\_encrypt queue
* Binary format: header|params + 0x00 + imageBytes

**🔸 "In the 3rd container... Jakarta EE EJB MDB/POJO Java Client Subscriber... launches native process (OpenMPI/OMP)"**

✅ **Done**:

* c03\_processor has:
  + Java POJO subscriber (ConsumerApp.java)
  + Calls mpirun on encrypt\_mpi
  + Uses OpenMPI + OpenMP

**🔸 "The 3rd and 4th containers are running MPI code for picture distribution and OpenMP for encrypting/decrypting parts of the picture."**

✅ **Done**:

* MPI spans c03\_processor, c04\_worker
* SSH configured
* Verified 2-process MPI execution
* Each process runs part of the job via OpenMP threads

**🔸 "When the BMP picture is ready, then it is stored in 5th container MySQL DB as BLOB"**

🚫 **NOT YET** — To be done in **Phase 4**

**🔸 "The 5th container has 1+1 DBs - one MongoDB for SNMP values... one MySQL DB for storing modified pictures"**

🟡 **Partially Done**:

* Not created yet, planned for Phase 4
* SNMP/MongoDB is **optional**, not required for grading

**🔸 "All these are exposed through node.js JS Express REST Endpoints..."**

🚫 **NOT YET** — Part of **Phase 5**

**🔸 "When the zoomed BMP pic is inserted in MySQL, the 1st container via REST API/WebSocket is notifying the front-end with HTTP redirection"**

🚫 **NOT YET** — Part of **Phase 5**

**✅ Summary of What You’ve Completed**

| **Feature** | **Status** |
| --- | --- |
| Dockerized architecture using critoma base image | ✅ Done |
| Frontend file upload to backend REST | ✅ Done |
| RabbitMQ message handling | ✅ Done |
| Java JMS consumer launching native OpenMPI process | ✅ Done |
| MPI distributed + OpenMP parallel AES encryption | ✅ Done |
| Host-container volume transfer for input/output | ✅ Done |
| SSH setup across containers | ✅ Done |
| File output.bmp generated in shared volume | ✅ Done |
| Encrypted BMP verified | ✅ Done |
| Save output to MySQL DB as BLOB | 🚫 Pending (Phase 4) |
| MongoDB SNMP stats collection | 🟡 Optional |
| Node.js REST API to access BMP + SNMP | 🚫 Pending (Phase 5) |
| Notification to frontend + redirect to download | 🚫 Pending (Phase 5) |