

Five containers, one cloud: **Mezarpoly** on Docker & Azure

A real-time multiplayer board game we built and designed as five containers: run locally with Docker Compose and deployed as five separate components on Azure Container Apps.

Azim

PART 1

Gloria

PART 2

Kutay

PART 3

The application and Docker

Azim
PRESENTER

- What Mezarpoly is and what the course asks for
- The architecture: five containers, two networks
- The containers and how their images are built

AGENDA

Three parts, three presenters

1 · Azim

The application and the requirements ·
the architecture · the containers and
their images

2 · Gloria

Docker Compose · the synchronous
path · Kafka and the worker ·
PostgreSQL · why Azure Container
Apps · the Azure architecture

3 · Kutay

The deploy script · security decisions ·
problems and fixes · proof that it works
· cost and next steps · questions

THE APPLICATION

Mezarpoly

Our game: real-time, browser-based, 2-8 players, live at mezarpoly.fun



- Roll dice, buy cities, build houses, trade, auction
- Live game over a **WebSocket**
- Accounts, shop, friends, leaderboards, ranked league
- Designed from day one as **five services**, switched on by configuration
- Public site: a light single-server mode was enough
- This project: the full **five-container** design

REQUIREMENTS

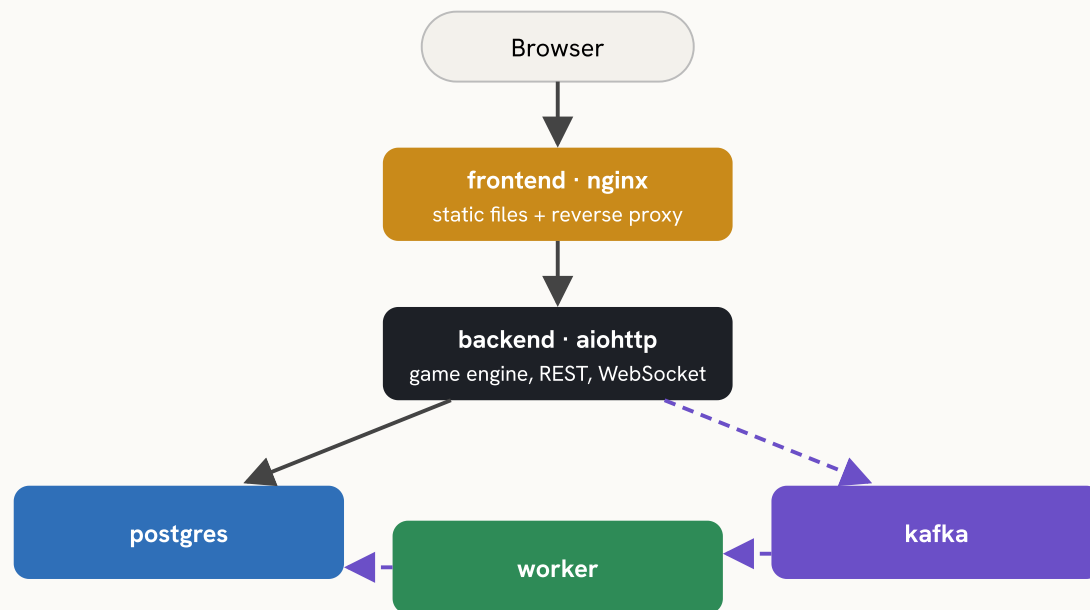
Grade map

Grade	Requirement	How it is met
3.0	Application starts with docker-compose.yml	<code>docker compose up --build</code> starts all five services
4.0	Two containers communicate over a Docker network	frontend (nginx) proxies HTTP + WebSocket to backend on network <code>web</code>
4.5	Third container: a database	PostgreSQL 17 on network <code>data</code> : accounts, results, shop, friends
5.0	Fourth container for asynchronous messaging	Kafka + worker: game events → Kafka → worker → PostgreSQL
6.0	All containers in the cloud as separate components	Azure Container Apps: one Container App per container

Course goal: package code and dependencies into containers that run the same in every environment, and build a small microservice-style system from them. Each grade's *result* is shown live: grades 3.0–5.0 on slide 11, grade 6.0 on slide 22.

ARCHITECTURE

Five containers, two networks



Network web

frontend, backend

Network data

backend, postgres, kafka, worker

Solid arrows: synchronous request/response.

Dashed arrows: asynchronous events.

The internet-facing frontend has **no route** to the database.

CONTAINERS

What runs where

Service	Image	Port	Role
frontend	own: Node 22 build → nginx 1.29	80	Static client, reverse proxy for /api /auth /ws
backend	own: Python 3.13 slim	8095	Rules engine, rooms, bots, accounts, REST, WebSocket, Kafka producer
postgres	postgres:17-alpine	5432	Relational database (19 tables)
kafka	apache/kafka:3.9.1 (KRaft)	9092	Message broker, topic <code>mezarpoly.events</code>
worker	own: Python 3.13 slim	—	Kafka consumer → PostgreSQL

Three images are built from this repository, two are official images. All four long-running services have health checks.

Three good habits in the Dockerfiles

Multi-stage build

The frontend compiles the Vue templates in a Node 22 stage, then copies only the result into nginx. No Node.js in the final image: smaller and safer.

Non-root user

backend and worker run as user `app` (uid 10001), not root. A compromised process gets very few rights inside the container.

Layer caching

Dependencies are copied and installed before the source code, so a code change rebuilds in seconds instead of reinstalling every library.

```
HEALTHCHECK --interval=10s --timeout=3s --retries=5 \  
  CMD python -c "import urllib.request; urllib.request.urlopen('http://127.0.0.1:8095/healthz', timeout=2)"
```