

Part 3:

Deployment, security, troubleshooting, proof

Your presenter guide: what to say on each of your slides, the background you need to answer questions confidently, likely questions with model answers, and a checklist for the day.

PRESENTER

Kutay

Your part at a glance

1 - Azim

Application, architecture,
containers, images · slides 1-8 · ~4
min

2 - Gloria

Compose, communication paths,
database, cloud platform · slides
9-16 · ~4 min

3 - Kutay

Deployment, security,
troubleshooting, proof, Q&A ·
slides 17-24 · ~4 min

You close the presentation. Gloria has shown the target architecture in Azure; you show **how it was built** (the deploy script), **how it is protected**, **what went wrong and how it was diagnosed**, the **evidence** that it runs end to end, the cost, and then you lead the questions.

Slide	Topic	Time
17	Part 3 divider	0:05
18	deploy.sh: seven idempotent steps	0:45
19	Security decisions	0:40
20	The "404 page not found" puzzle	0:55
21	Other problems on the way	0:30
22	Evidence: it runs end to end	0:35
23	Cost and next steps	0:30
24	Summary, thank you, questions	0:20
	Total (before Q&A)	≈ 4:15

Your three key messages

- The whole Azure environment is created by **one reproducible script** (infrastructure as code).
- Security by design: **one public endpoint**, secrets, managed identity, least privilege.
- We **diagnosed problems with evidence** — the 404 story shows real understanding of how Container Apps routes traffic.

How you take over and close

Gloria ends with: "Kutay will now show how we actually built it...". Start with: *"Thank you, Gloria."* When Gloria stops sharing, share your **whole screen** (not a single window): you switch between Slides-Part3-Kutay.pdf, the browser with the game and status page, and the Azure portal during the demo. On the last slide say *"Thank you — we are happy to take questions"* and invite your teammates to answer questions about their parts.

Part 3 divider — Deployment, security, troubleshooting, proof

PART 3 OF 3 · ABOUT 4 MINUTES

Deployment, security, troubleshooting, proof

Kutay
PRESENTER

- The deploy script, step by step
- Security decisions
- Problems we hit and how we fixed them
- Evidence that it works, cost and next steps

Mezarpoly · Docker & Azure

Part 3

WHAT TO SAY

Thank you, Gloria. I will show how we built this environment, how it is protected, what went wrong, and the proof that it works.

deploy.sh — seven idempotent steps

INFRASTRUCTURE AS CODE

PART 3 - KUTAY

deploy.sh: seven idempotent steps

1. Resource group + container registry
2. VNet + subnet delegated to `Microsoft.App`
3. Container Apps environment in that subnet
4. postgres — internal TCP 5432, password as a secret
5. kafka — internal TCP 9092, single-node KRaft
6. worker (no ingress) + backend (internal HTTP)
7. frontend — external HTTPS, then tell the backend its public origin

```
az containerapp create -g mezarpoly-rg -n postgres \
  --environment mezarpoly-env \
  --registry-identity system --image $REG/postgres:17-
alpine \
  --secrets pgpass="$PG_PASS" \
  --env-vars POSTGRES_PASSWORD=secretref:pgpass \
  --ingress internal --transport tcp \
  --target-port 5432 --exposed-port 5432
```

Mezarpoly · Docker & Azure

18

WHAT TO SAY

We did not click the environment together in the Azure portal. One script, using the Azure CLI, creates everything — so the script is also the documentation, and anyone can reproduce the environment.

It is **idempotent**: if you run it again, it skips what already exists and only rolls out new image versions.

The seven steps are: a resource group and a container registry; a virtual network with a subnet delegated to Container Apps; the Container Apps environment inside that subnet; then PostgreSQL with internal TCP on port 5432, Kafka with internal TCP on 9092, the worker without ingress and the backend with internal HTTP; and finally the frontend with external HTTPS — after which we tell the backend its public address.

The example on the right creates the database app: the image is pulled with the app's **managed identity**, the password is stored as a **secret**, and the ingress is internal TCP on 5432.

REMEMBER

- Idempotent = safe to run again.
- Infrastructure as code = reproducible + documented.

Security decisions

SECURITY

PART 3 - KUTAY

Decisions that reduce risk

Small attack surface

One external endpoint (HTTPS, certificate by Azure). Database and Kafka are unreachable from the internet.

Secrets, not plain text

Random passwords generated at deploy time, stored as Container App secrets, injected with `secretref:`.

No registry password

Apps pull images with a system-assigned managed identity that has pull rights on the registry.

Least privilege

Non-root containers; the frontend has no network path to data.

CSRF protection

The backend accepts POST and WebSocket only from its own public origin.

Real client IP

nginx forwards the first `X-Forwarded-For` address so rate limits see the user, not the load balancer.

Mezarpoly · Docker & Azure

19

WHAT TO SAY

Six decisions reduce the risk.

The most important: there is exactly **one public endpoint**, the frontend, with HTTPS and a certificate managed by Azure. The database and Kafka cannot be reached from the internet at all.

Passwords are generated randomly when we deploy, stored as Container App **secrets** and referenced with `secretref`, so they never appear in plain text in the script or in the portal.

There is no registry password either: every app authenticates to the registry with a **system-assigned managed identity**.

The containers run as non-root, and the frontend has no network path to the data.

The application keeps its own protections: the backend only accepts POST and WebSocket requests from its own public origin — that is **CSRF** protection — and nginx forwards the real client IP from `X-Forwarded-For`, so per-IP rate limits see the user and not Azure's load balancer.

REMEMBER

- One public endpoint · secrets · managed identity · non-root · CSRF origin check · real client IP.

The "404 page not found" puzzle

TROUBLESHOOTING

PART 3 - KUTAY

The "404 page not found" puzzle

Symptom: the home page loaded, but `/healthz`, `/api` and the WebSocket returned `404 page not found`.

Diagnosis: that text is the environment's internal router (Envoy), not nginx. nginx forwarded to `http://backend`; internal HTTP traffic is routed by Host header and only the app's **full internal name** matches.

Fix:
`BACKEND_URL=http://backend.internal.<env domain>`, computed in `deploy.sh` so it never comes back.

Why did postgres:5432 work?
TCP ingress is forwarded directly, so the short name resolves. HTTP ingress goes through the router, which matches host names. Same environment, two behaviours.

Mezarpoly · Docker & Azure

20

WHAT TO SAY

This was our most instructive problem. After the first deployment the home page loaded — so nginx was fine — but every call to the backend, the health check, the API and the WebSocket, returned "404 page not found".

The wording gave it away: that is not nginx's error page. It comes from **Envoy**, the internal router of the Container Apps environment. So nginx was forwarding the request, but the router did not know where to send it.

Locally, nginx sends traffic to `http://backend` and Docker's DNS turns "backend" into the container. In Azure, internal **HTTP** traffic goes through the router, which chooses the destination by the **Host header** — and it only recognises the app's **full internal name**: `backend.internal.` followed by the environment's domain.

Why did `postgres:5432` work with the short name? Because **TCP** ingress is forwarded directly, without host-name routing. Same environment, two behaviours.

We fixed it by reading the environment's default domain in the deploy script and setting `BACKEND_URL` to the full internal name, so a fresh deployment works the first time.

REMEMBER

- Clue: the error text was Envoy's, not nginx's.
- HTTP ingress → routed by Host header → needs full internal FQDN.
- TCP ingress → forwarded directly → short name works.

Other problems on the way

TROUBLESHOOTING
PART 3 - KUTAY

Other problems on the way

Problem	Cause	Fix
Container Apps commands rejected	Resource provider <code>Microsoft.App</code> not registered	<code>az provider register</code> , wait for <i>Registered</i>
Resources refused in some regions	Azure for Students region policy	France Central
No TCP ingress	Requires a custom VNet	VNet + delegated /23 subnet
Logins and moves rejected	Origin check did not know the new public URL	Set <code>MP_ORIGINS</code> after the frontend exists
<code>GroupCoordinatorNotAvailable</code> in worker logs	Kafka still creating internal topics	Harmless; retries until "listening"

Mezarpoly · Docker & Azure
21

WHAT TO SAY

Some smaller problems, each with a clear cause.

Container Apps commands were first rejected because the resource provider `Microsoft.App` was not registered in our student subscription; we registered it and waited until it said "Registered".

The student subscription only allows certain regions, so we used France Central.

TCP ingress required the custom virtual network, as Gloria explained.

Logins and moves were rejected until the backend learned its new public URL — that is the origin check. And the `GroupCoordinatorNotAvailable` message in the worker logs looks scary but is harmless: Kafka is still creating its internal topics, and the worker retries until it is listening.

REMEMBER

- Name the cause, not only the fix — that shows understanding.

Evidence — it runs end to end on Azure

EVIDENCE

It runs end to end on Azure

PART 3 - KUTAY



App	State	External	Transport
postgres	Running	False	Tcp
kafka	Running	False	Tcp
worker	Running		
backend	Running	False	Http
frontend	Running	True	Auto

[events] connected to Kafka: kafka:9092

[worker] listening: kafka:9092 / mezarpoly.events

[worker] processed: game_started 2RYA9

Mezarpoly - Docker & Azure

22

WHAT TO SAY

Here is the evidence. On the left, the game served from the Azure public URL.

On the right, the five Container Apps: only the frontend is external, PostgreSQL and Kafka use TCP, the backend uses internal HTTP.

And the logs of a real game: the backend connected to Kafka, the worker is listening, and the last line — `processed: game_started 2RYA9` — shows that a game started in room 2RYA9, the backend published the event, and the worker wrote it into PostgreSQL.

That single line proves the asynchronous path and the separate cloud components working together.

REMEMBER

- Point at the last log line when you say "that single line".

Cost, and what we would do next

OPERATIONS

PART 3 - KUTAY

Cost, and what we would do next

Cost control

- `stop.sh` deactivates every revision → 0 replicas, no compute cost
- `start.sh` brings everything back in 1-2 minutes
- Registry (Basic) is the only fixed cost
- Data in postgres/kafka is ephemeral in this setup

Next steps

- Managed PostgreSQL + Event Hubs (Kafka API) for durable data
- CI/CD: GitHub Actions builds and deploys on every push
- Central logs and metrics with Log Analytics
- Shared room state (Redis) to scale the backend horizontally

Mezarpoly · Docker & Azure

23

WHAT TO SAY

Running five apps all day would use up the student credit, so two scripts stop and start the whole system. `stop.sh` deactivates every revision, which means zero replicas and no compute cost; `start.sh` brings everything back in one to two minutes. The registry is the only fixed cost.

We are honest about the limits: data in PostgreSQL and Kafka is not persistent in this setup.

For production we would use Azure's managed PostgreSQL and Event Hubs, which speaks the Kafka protocol; add a CI/CD pipeline that deploys on every push; collect logs centrally; and move the room state to Redis so the backend could scale to several replicas.

REMEMBER

- Mentioning limits yourself earns credibility.

Summary — thank you, questions

SUMMARY

Thank you Questions?

- Five containers, two networks, one command locally
- Synchronous path through nginx; asynchronous events through Kafka
- Five separate Container Apps on Azure, one public endpoint
- Reproducible with one script; debugged with evidence

Azim · Gloria · Kutay 24

WHAT TO SAY

To summarise: our game in its full five-container design, on two networks, started with one command; a synchronous path through nginx and an asynchronous path through Kafka; and five separate Container Apps on Azure with one public endpoint, built by one reproducible script.

Thank you. We are happy to take your questions — Azim and Gloria will answer questions about their parts, and we can show the running system or the scripts live.

REMEMBER

- If a question is about Docker images → Azim; Compose/Kafka/database → Gloria; Azure/deployment → you.

Background you should understand

Azure concepts

Concept	Meaning	Ours
Subscription	Billing account that owns resources	Azure for Students
Resource group	Folder for related resources	mezarpoly-rg
Region	Data-centre location	France Central
Resource provider	Enables a service type in a subscription	Microsoft.App
Container Registry	Private image store	ACR Basic
VNet / subnet	Private network; subnet delegated to Container Apps ($\geq$ /23)	10.60.0.0/16, apps 10.60.0.0/23
Environment	Boundary where apps live and find each other	mezarpoly-env
Container App	One container with its ingress, scale and secrets	5 apps
Revision	Immutable snapshot of an app's image + settings	new one on every change
Replica	A running instance of a revision	min = max = 1

The deploy script — key commands

```
az group create -n mezarpoly-rg -l francecentral
az acr create -g mezarpoly-rg -n <registry> --sku Basic
az network vnet create ... --address-prefixes 10.60.0.0/16
az network vnet subnet create ... --address-prefixes 10.60.0.0/23 \
  --delegations Microsoft.App/environments
az containerapp env create ... --infrastructure-subnet-resource-id $SUBNET_ID \
  --enable-workload-profiles
az containerapp create -n backend ... --ingress internal --transport http --target-port 8095
az containerapp create -n frontend ... --ingress external --target-port 80
DOMAIN=$(az containerapp env show ... --query properties.defaultDomain -o tsv)
BACKEND_URL="http://backend.internal.$DOMAIN"
```

Ingress types

- **External** — reachable from the internet; Azure terminates HTTPS with its own certificate (frontend).
- **Internal HTTP** — reachable only inside the environment, routed by host name through Envoy (backend).
- **Internal TCP** — reachable only inside the environment on a fixed port, forwarded directly (postgres 5432, kafka 9092). Requires a custom VNet.
- **None** — not reachable at all; only outgoing connections (worker).

Secrets and managed identity

A **secret** is stored on the Container App and injected into an environment variable with `secretref:name`; it is not shown in the portal or written to logs. Passwords are created at deploy time with `openssl rand -hex 16`. A **managed identity** is an identity Azure gives to a resource; we gave it pull rights on the registry, so no registry username or password exists anywhere.

CSRF and the origin check

Cross-Site Request Forgery: a malicious page makes the browser send a request to our site with the user's cookies. The backend refuses POST and WebSocket requests whose `Origin` header is not its own public URL. In Azure that URL is only known after the frontend is created, so the script sets `MP_ORIGINS`, `MP_PUBLIC_URL` and `MP_API_URL` at the end.

Real client IP behind two proxies

Traffic passes Azure's load balancer and then nginx. The backend would see a proxy IP; rate limits and anti-abuse rules would treat all users as one. Azure writes the real address into `X-Forwarded-For`; nginx takes the first address and passes it on.

Operating the system

```
az containerapp list -g mezarpoly-rg -o table
az containerapp logs show -g mezarpoly-rg -n worker --tail 50
az containerapp revision list -g mezarpoly-rg -n backend
bash docker/azure/stop.sh      # deactivate revisions → 0 replicas
bash docker/azure/start.sh     # back in 1-2 minutes
az group delete -n mezarpoly-rg # delete everything
```

Start the system **at least 10 minutes before** the presentation if you want to show it live.

Questions you may get — with model answers

You close the talk and lead the Q&A, so most technical questions about Azure will come to you. Answer each one out loud, then compare.

1. Why Azure Container Apps and not Kubernetes?

We need separate components with internal networking and TCP support, but not the effort of running a Kubernetes cluster. Container Apps runs on Kubernetes that Microsoft manages, so we get the benefits without operating it.

2. What does idempotent mean for your script?

Running it several times gives the same result: existing resources are skipped or updated, nothing is duplicated, and new image tags are rolled out.

3. How are secrets handled?

Generated randomly at deploy time, stored as Container App secrets, referenced with secretref in environment variables. They are not in the repository, the script output or the portal view.

4. How do the apps pull images without a password?

Each app has a system-assigned managed identity with pull permission on the registry.

5. Explain the 404 problem in one minute.

nginx forwarded to http://backend. Internal HTTP traffic in Container Apps is routed by Envoy using the Host header, and only the full internal name backend.internal. matches; the short name returned Envoy's 404. TCP ingress (postgres, kafka) is forwarded directly, so short names work there. We compute the full name from the environment's default domain in deploy.sh.

6. How did you know the 404 came from Azure and not from nginx?

The response body "404 page not found" is Envoy's plain-text page; nginx returns an HTML error page. Also, static files from nginx loaded fine, so nginx itself was working.

7. Why is there only one backend replica?

Game rooms are kept in the backend's memory; two replicas would each have their own rooms and WebSocket connections. To scale out, room state would have to move to a shared store such as Redis.

8. Is the data persistent in Azure?

Not in this setup: postgres and kafka have no persistent volumes, so a restart resets them. For production we would use Azure Database for PostgreSQL and Event Hubs.

9. How much does it cost?

While running, the apps are billed per vCPU and memory second; with stop.sh all revisions are deactivated, so there is no compute cost. The Basic registry has a small daily fixed cost. Everything can be removed with `az group delete`.

10. How did you prove it works?

We played a game through the public URL; the worker log shows "processed: game_started 2RYA9", which means the event went backend → Kafka → worker → PostgreSQL.

11. What would you improve first?

Persistent managed data services, then a CI/CD pipeline with GitHub Actions, then central logging and monitoring with Log Analytics.

12. What is a revision?

An immutable snapshot of a Container App's image and settings. Every change creates a new revision; you can roll back, and stop.sh works by deactivating revisions.

Terms in your part

Infrastructure as code

Creating infrastructure from scripts instead of clicks.

Azure CLI (az)

Command-line tool for Azure.

Idempotent

Same result however often you run it.

Resource provider

Enables a service type (Microsoft.App) in a subscription.

Delegated subnet

Subnet handed to a service (Container Apps).

Environment

Container Apps boundary; apps inside share a network.

Ingress

How an app is reached: external, internal HTTP, internal TCP, none.

Envoy

Proxy that routes internal HTTP traffic in Container Apps.

FQDN

Fully qualified domain name, e.g. backend.internal..

Secret / secretref

Hidden value and the reference that injects it.

Managed identity

Password-less identity Azure assigns to a resource.

Revision / replica

Immutable version of an app / running instance of it.

CSRF

Attack that sends requests with the user's cookies from another site.

X-Forwarded-For

Header carrying the real client IP through proxies.

Checklist before you present

- ☐ Share your whole screen, not one window; open Slides-Part3-Kutay.pdf, the game, /status.html and the Azure portal in advance.
- ☐ Rehearse your part out loud twice with a timer (target: about 4 minutes).
- ☐ Be able to tell the 404 story in under a minute without the slide.
- ☐ If you show the system live: run start.sh at least 10 minutes before; open the public URL and /status.html in two tabs.
- ☐ Have az containerapp logs show ... -n worker ready in a terminal.
- ☐ Have the screenshots as a backup if the network fails.
- ☐ Run stop.sh after the presentation.
- ☐ Agree with Azim and Gloria who answers which questions.