

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

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
```

SECURITY

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.

TROUBLESHOOTING

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?

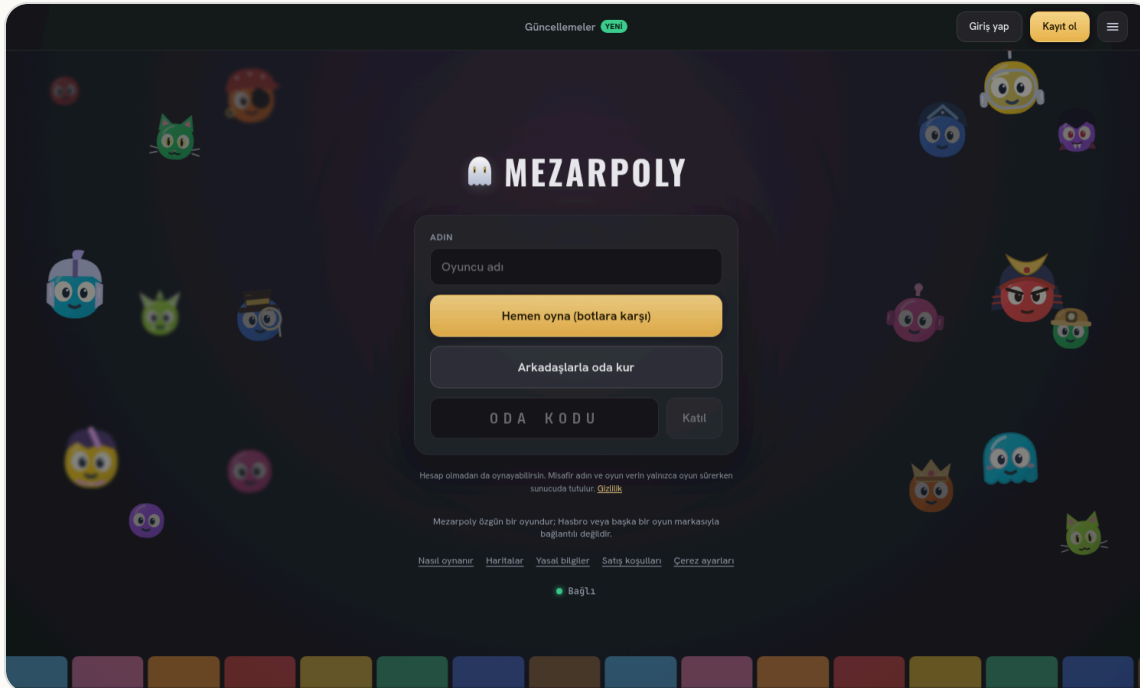
TCP ingress is forwarded directly, so the short name resolves. HTTP ingress goes through the router, which matches host names. Same environment, two behaviours.

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"

EVIDENCE

It runs end to end on Azure



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
```

OPERATIONS

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

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