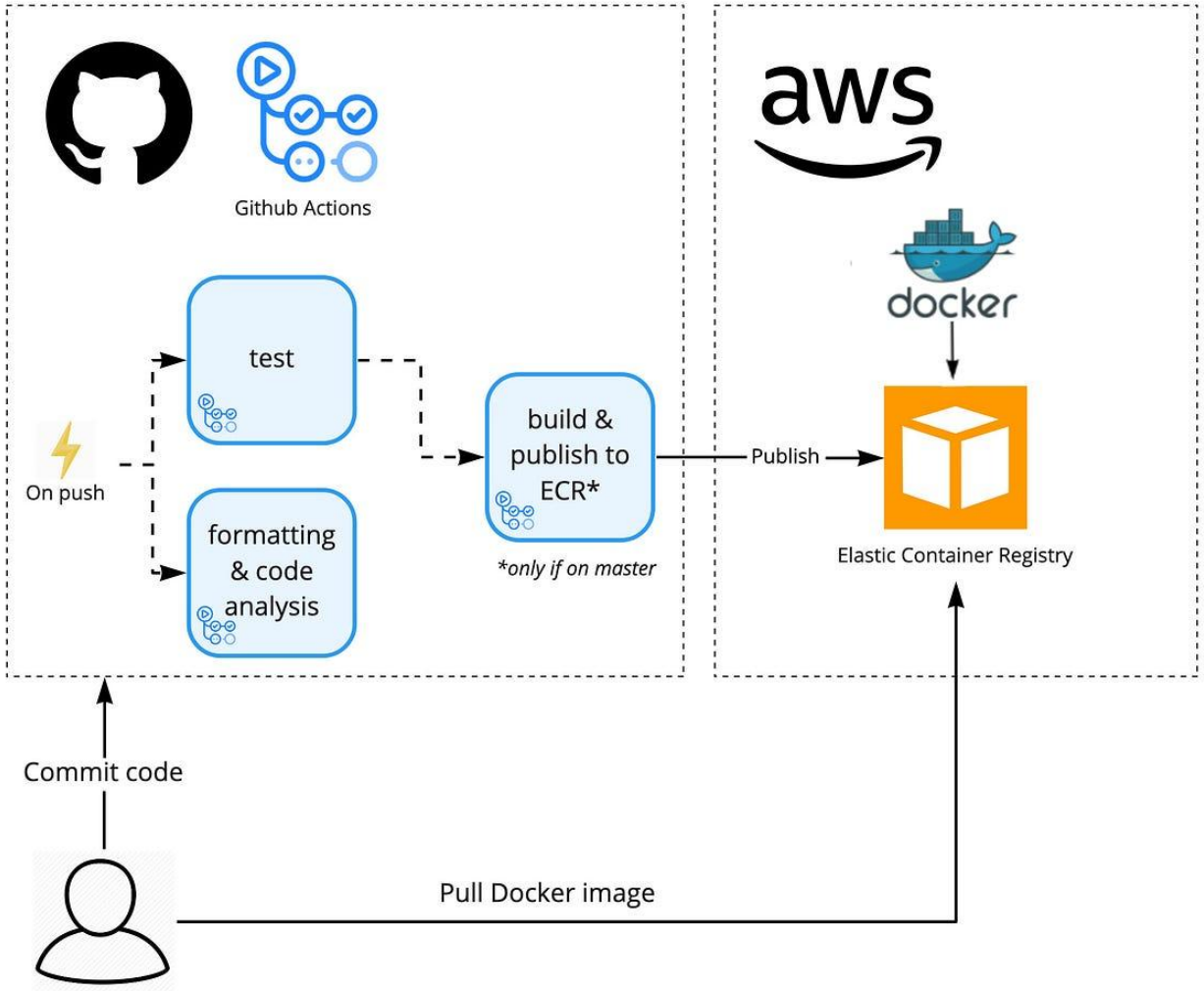
МОНГОЛ УЛСЫН ИХ СУРГУУЛЬ
Хэрэглээний шинжлэх ухаан, Инженерчлэлийн сургууль
Мэдээлэл, компьютерийн ухааны тэнхим

Week 4
End-to-End Deployment Workshop

Advanced Web Application Development
National University of Mongolia | Fall 2025
Lecturer R.Javkhlan

Today's Goals

- A live HTTPS URL serving Yellow Books (Next.js web + API).
- CI/CD pipeline: GitHub Actions → ECR → EKS (no static AWS keys).
- Kubernetes resources: Deployments, Services, Ingress (ALB), Secrets/ConfigMaps, Migration Job, and HPA.
- A repeatable DEPLOY.md you can reuse for future projects.



Reading Map (Modules & Time)

- **Module 0: Setup & Repo**
- **Module 1 — Docker & ECR + GitHub Actions (Build & Push Now)**
- Module 2: Kubernetes & EKS
- Module 3: Ingress, TLS, DNS
- Module 4: IAM, OIDC, IRSA
- Module 5: K8s Manifests + Prisma Job
- Module 6: Verify, Troubleshoot, Teardown

Module 0 — Repo & Local Hygiene

Create/Prepare GitHub Repo

1. Create a private repo (name idea: yellowbooks-eks).
2. Initialize git locally, commit, and push to main.
3. Set up SSH or HTTPS auth (SSH recommended).
4. Turn on branch protection for main.
5. Sanity-check with two commands.

Steps. Create repo on GitHub

1. Web UI: GitHub → New → Private → Name:
   yellowbooks-eks → Add README ✔️ → Create.
2. Or CLI:

```
# from your project folder (containing apps/, libs/, prisma/ ...)
gh repo create <ORG/REPO> --private --source=. --remote=origin --push
```

3. Configure git (first time only)

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

Steps. Add remote & push main

```
# If repo was not created with gh CLI
git init -b main
git add .
git commit -m "chore: initial commit (apps/web, apps/api, libs, prisma,
# choose ONE remote style
# SSH (recommended)
git remote add origin git@github.com:<ORG/REPO>.git
# or HTTPS
git remote add origin https://github.com/<ORG/REPO>.git

git push -u origin main
```

Steps. Enable branch protection (UI)

Settings → Branches → Branch protection rules → Add rule:

- Branch name pattern: main

✔️ Require a pull request before merging (optional for class)

✔️ Require status checks to pass before merging (we'll wire CI soon)

✔️ Do not allow bypassing the above settings

- Save.
- Verify

```
git remote -v      # should show origin → github.com/<ORG/REPO>
git branch -vv     # should show * main tracking origin/main
```

Project Layout

```
apps/
  web/              # Next.js (App Router)
  api/              # Fastify/Express (REST for yellow-books)
libs/
  contract/         # Zod schemas + TypeScript types shared by web/api
  config/           # Env loader (zod-validated), no secrets
prisma/             # schema.prisma + migrations/ + seed script
k8s/                # Kubernetes manifests (deployments, services, ingress
.github/workflows/# GitHub Actions (CI/CD)
```

Minimum required files

- apps/web/package.json → build script produces Next.js standalone output.
- apps/api/package.json → build compiles to apps/api/dist; start runs node dist/main.js.
- libs/contract/index.ts exports YellowBookEntrySchema + YellowBookEntry type.
- prisma/schema.prisma has YellowBookEntry model; prisma/seed.ts seeds ≥5 records.
- k8s/ contains:
  - api-deploy.yaml, web-deploy.yaml (Deployments + Services)
  - ingress-http.yaml and ingress-https.yaml (ALB + optional TLS)
  - job-migrate.yaml (Prisma migrate deploy)
  - hpa.yaml (API autoscaling)
- .github/workflows/cicd.yml (build, push, deploy).

Helpful root scripts (optional but nice)

```
// package.json (root)
{
  "scripts": {
    "build:web": "pnpm -C apps/web build",
    "build:api": "pnpm -C apps/api build",
    "lint": "nx run-many -t lint -p web api",
    "typecheck": "tsc -b --noEmit"
  }
}
```

- Separation of concerns: web vs api code paths are clean → easier Dockerfiles.
- Shared contracts: one source of truth for data shapes → fewer runtime bugs.
- Infra as code: k8s/ checked in → reproducible deployments.
- CI discoverability: workflows live in standard location so GitHub auto-detects.

.dockerignore & Build Context

- **Why it matters:** Docker sends your build context (the files under .) to the daemon.
- Big contexts → slow builds, cache misses, higher CI cost.
- **Add this at repo root:**

```
# node & package managers
node_modules
pnpm-lock.yaml#ignore? (NO – keep for reproducible installs)

# VCS & artifacts
.git
*.log
coverage

# Next.js & builds
.next
out
dist

# tooling & caches
pnpm-store
.cache

# tests & storybook outputs (optional)
cypress
storybook-static

# env files (never commit secrets)
.env
.env.*
```

Module 1 — Docker & ECR + GitHub Actions (Build & Push Now)

Docker 101 (Image vs Container · Deep Dive)

Why Docker? Package once → run anywhere (your laptop, CI, Kubernetes).

- Image = read-only blueprint (layers). Built from a Dockerfile.
- Container = running instance of an image (ephemeral).
- Layers & cache: Each RUN/COPY/ADD creates a cacheable layer → order your Dockerfile to maximize reuse.
- Multi-stage builds: Build in a full toolchain image → copy only compiled artifacts to a tiny runtime image.
- Base images: node:20-alpine (small), node:20-bookworm-slim (more libs, fewer native module issues), distroless (very small, advanced).
- OCI: Docker images follow the OCI standard → works with many registries (ECR, GHCR, Docker Hub).

Dockerfile anatomy

- FROM base → WORKDIR → COPY → RUN (install/build) → ENV → EXPOSE → CMD or ENTRYPOINT.

Common pitfalls

- Copying the entire repo into the **runtime** stage → huge images.
- Putting secrets in ENV during build → secrets leak into image history.

```
# 1. Choose base image
FROM node:18

# 2. Set working directory inside the container
WORKDIR /app

# 3. Copy files from your computer into the container
COPY package*.json ./
RUN npm install

# 4. Copy the rest of the code
COPY . .

# 5. Define how to start your app
CMD ["npm", "start"]

# 6. Expose the port (optional)
EXPOSE 3000
```

```
FROM node:20-bookworm-slim AS deps
WORKDIR /app
COPY package.json pnpm-lock.yaml ./
RUN corepack enable && corepack prepare pnpm@9 --activate && pnpm fetch
COPY . .
RUN pnpm i --offline && pnpm build

FROM node:20-bookworm-slim AS runtime
WORKDIR /app
COPY --from=deps /app/dist ./dist
RUN useradd -m -u 10001 nodeuser
USER nodeuser
EXPOSE 3000
CMD ["node","dist/main.js"]
```

```
# inventory & layers
docker images && docker ps -a
docker history yb-api:local  # see which steps are big
# clean up
docker rm -f $(docker ps -aq)   # remove all containers (careful!)
docker rmi $(docker images -q)  # remove all images (careful!)
```

Registries 101 & Amazon ECR (Tags · Scanning · Lifecycle · Cross-Region)

Registry = storage for versioned images. We use Amazon ECR (private, regional, IAM-secured).

- URI format: <acct>.dkr.ecr.<region>.amazonaws.com/yellowbooks/api:<tag>.

Tagging strategy

- Immutable: commit SHA (e.g., :3a1b2c4).
- Channel: :main, :staging, :pr-123 (optional extra tag).
- For releases: :v1.2.0 in addition to SHA.
- Prefer pinning by digest or SHA in deploys.

Security & hygiene

- Enable Scan on push (detect CVEs in base images).
- Add Lifecycle policy (e.g., keep last 20 tags) to control storage cost.
- By default, ECR images are encrypted at rest (KMS).

Manual login & push

Keep **ECR and EKS in the same region** to avoid pull latency and permission surprises.

```
aws ecr get-login-password --region <REGION> | \
docker login --username AWS --password-stdin <ACCT>.dkr.ecr.<REGION>.amazonaws.com
docker tag yb-api:local <ACCT>.dkr.ecr.<REGION>.amazonaws.com/yellowbooks/api:test
docker push <ACCT>.dkr.ecr.<REGION>.amazonaws.com/yellowbooks/api:test
```

```
{
    "rules": [
        {
        "rulePriority": 1,
        "description": "Keep last 20 images (any tag)",
        "selection": { "tagStatus": "any", "countType": "imageCountMoreThan",
        "countNumber": 20 },
        "action": { "type": "expire" }
        }
    ]
}
```

API Dockerfile (Multi-Stage · Fast Rebuilds · Secure Runtime)

- **Goals:** reproducible deps, small image, non-root, fast CI.
- **Key choices:** bookworm-slim base; pnpm workspace; copy only compiled output.

```dockerfile
# apps/api/Dockerfile
# syntax=docker/dockerfile:1.6
FROM node:20-bookworm-slim AS base
WORKDIR /app

# deps – cache on lockfile
FROM base AS deps
RUN corepack enable && corepack prepare pnpm@9 --activate
COPY pnpm-workspace.yaml package.json pnpm-lock.yaml ./
COPY apps/api/package.json apps/api/package.json
COPY libs/contract/package.json libs/contract/package.json
RUN pnpm fetch --prod
COPY . .
RUN pnpm -C apps/api install --offline --prod=false

# build – compile TS → JS
FROM deps AS build
RUN pnpm -C apps/api build  # emits apps/api/dist

# runtime – minimal files + non-root
FROM node:20-bookworm-slim AS runtime
ENV NODE_ENV=production
WORKDIR /app
COPY --from=deps /app/apps/api/package.json ./package.json
RUN corepack enable && corepack prepare pnpm@9 --activate \
 && pnpm i --prod --filter ./
COPY --from=build /app/apps/api/dist ./dist
RUN useradd -m -u 10001 nodeuser
USER nodeuser
EXPOSE 3000
CMD ["node","dist/main.js"]
```

Web Dockerfile (Next.js Standalone · Build-time vs Runtime Env)

- **Goals:** small server image that serves SSR/ISR; keep secrets out of build.
- **Next.js config:** ensure next.config.js sets output: 'standalone'.

```
# apps/web/Dockerfile
# syntax=docker/dockerfile:1.6
FROM node:20-bookworm-slim AS base
WORKDIR /app

FROM base AS deps
RUN corepack enable && corepack prepare pnpm@9 --activate
COPY pnpm-workspace.yaml package.json pnpm-lock.yaml ./
COPY apps/web/package.json apps/web/package.json
COPY libs/contract/package.json libs/contract/package.json
RUN pnpm fetch --prod
COPY . .
RUN pnpm -C apps/web install --offline --prod=false

FROM deps AS build
ENV NEXT_TELEMETRY_DISABLED=1
# Only NEXT_PUBLIC_* is safe at build — server secrets must be runtime
RUN pnpm -C apps/web build  # emits .next/standalone + .next/static

FROM node:20-bookworm-slim AS runtime
ENV NODE_ENV=production PORT=3000 HOSTNAME=0.0.0.0
WORKDIR /app
COPY --from=build /app/apps/web/.next/standalone ./
COPY --from=build /app/apps/web/public ./public
COPY --from=build /app/apps/web/.next/static ./.next/static
RUN useradd -m -u 10002 nextuser
USER nextuser
EXPOSE 3000
CMD ["node","server.js"]
```

Local Sanity Checklist (Run Images Before AWS)

```
export DATABASE_URL=postgres://user:pass@localhost:5432/yellow

docker build -f apps/api/Dockerfile -t yb-api:local .
docker run --rm -p 3000:3000 -e DATABASE_URL yb-api:local
# new terminal
curl -s http://localhost:3000/health && echo
```

Typical issues & quick fixes

- API 500 on start → wrong/missing DATABASE_URL; DB not reachable.
- Web cannot load data → wrong NEXT_PUBLIC_API (use http://localhost:3000).
- Port already in use → stop other processes or change -p mapping.
- CORS (if browser calls API) → allow origin http://localhost:3001 in API for local test.

```
docker build -f apps/web/Dockerfile -t yb-web:local .
docker run --rm -p 3001:3000 -e NEXT_PUBLIC_API=http://localhost:3000
open http://localhost:3001/yellow-books
```

Success checklist

- /health returns 200 < 100ms.
- /yellow-books shows ≥ 5 records.
- docker history shows sane size (< ~300MB for class is fine).

Why Move CI Early? (Aha Now, Details Later)

Pros

- Instant feedback loop: push ⇒ build ⇒ images in ECR.
- Mirrors real teams: CI produces artifacts on every commit.
- Motivates proper Dockerfiles and reproducible builds.

Cons

- YAML, secrets, permissions can be overwhelming → we limit scope to ECR push only this week.
- OIDC and cluster permissions not covered yet → we use a temporary IAM user only for ECR (delete next week).

Add GitHub Secrets & Variables (UI Walkthrough)

Where: GitHub → Repo → Settings → Secrets and variables → Actions.

Add these:

- Secrets: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY.
- Variables: AWS_REGION, ECR_REPO=<acct>.dkr.ecr.<region>.amazonaws.com/yellowbooks, DOCKER_BUILDKIT=1.

Tips:

- Use repo-level secrets for class; org-level if many repos.
- Masked by default; rotate if exposed.
- You can create environments (staging/prod) later with separate secrets.

Actions Workflow (Build & Push Only — No Deploy Yet, with Caching)

- .github/workflows/ci-build-push.yml (annotated):
- https://drive.google.com/file/d/1cFS2QIS6ubozD6O8jMcQtE-Drj_47mwo/view?usp=sharing

What this does

- Restores pnpm cache → faster builds.
- Uses buildx with GitHub cache for Docker layers.
- Pushes SHA tag every time; on main, also pushes :main tag.

Nice-to-Have: Status Badge & PR Checks

Badge (README.md):
![CI](https://github.com/<ORG/REPO>/actions/workflows/ci-build-push.yml/badge.svg)

Require check on PRs:

- Settings → Branches → Branch protection → Require status checks to pass → select ci-build-push.

Nice-to-Have: Status Badge & PR Checks

Badge (README.md):
![CI](https://github.com/<ORG/REPO>/actions/workflows/ci-build-push.yml/badge.svg)

Require check on PRs:

- Settings → Branches → Branch protection → Require status checks to pass → select ci-build-push.

Homework & Rubric & Deliverables

- **Deliverables:** repo link, CI run link (green), ECR screenshots (both images with :<sha>), updated README badge.
- **Rubric (100):** Dockerfiles 30 · Local sanity 10 · ECR repos+policies 20 · CI build/push 30 · Docs 10.
- **Bonus (+10 means 1 point):** matrix build for push and pull_request.

АСУУЛТ ?