Full Tutorial: Docker, NodeJS, MySQL, EJS Frontend, and Kubernetes

This tutorial explains a full-stack application setup using Docker, NodeJS, MySQL, EJS frontend, and Kubernetes.
Everything, including explanations, key concepts, bullet points, and code examples, is fully contained in this Markdown file.

Table of Contents

Dockerfiles

Backend Dockerfile

```
FROM node:20-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
ENV PORT=3001
EXPOSE 3001
CMD ["sh", "-c", "node ./utils/wait-for-db.js && node ./bin/www"]
```

Key Concepts

- `FROM node:20-alpine` → lightweight Node.js image
- `WORKDIR` → sets the working directory inside the container
- `COPY package*.json ./` + `RUN npm ci --omit=dev` → installs dependencies
- `COPY . .` → copies application code
- `ENV PORT=3001` → internal environment variable
- `EXPOSE 3001` → internal listening port
- `CMD` → runs API server after waiting for MySQL

Frontend Dockerfile

```
FROM node:20-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
ENV PORT=3000
EXPOSE 3000
CMD ["node", "./bin/www"]
```

Key Concepts

- Similar structure to backend
- `EXPOSE 3000` → internal port only
- Omitting `ports:` in Compose keeps frontend internal

Docker Compose

```
version: "3.9"

services:
    db:
        image: mysql:8.0
        environment:
            MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD:-rootpass}
            MYSQL_DATABASE: ${MYSQL_DATABASE:-todos}
            MYSQL_USER: ${MYSQL_USER:-todo_user}
            MYSQL_PASSWORD: ${MYSQL_PASSWORD:-todo_pass}
        ports:
            - "3306:3306"
        volumes:
            - db_data:/var/lib/mysql
        networks:
            - appnet

    api:
        build: ./backend
        depends_on:
            db:
                condition: service_healthy
        environment:
            DB_HOST: db
            DB_USER: ${MYSQL_USER:-todo_user}
            DB_PASS: ${MYSQL_PASSWORD:-todo_pass}
        ports:
            - "3001:3001"
        networks:
            - appnet

    web:
        build: ./frontend
        environment:
            API_BASE_URL: "http://api:3001"
        networks:
            - appnet

volumes:
    db_data:

networks:
    appnet:
        driver: bridge
```

Key Concepts

- `services` → defines each container (db, api, web)
- `depends_on` → ensures startup order (API waits for DB)
- `volumes` → persistent storage for MySQL data
- `networks` → allows containers to communicate using service names
- `ports` → maps container ports to host; omit to keep service internal

---

Environment Variables

```
MYSQL_ROOT_PASSWORD=rootpass
MYSQL_DATABASE=todos
MYSQL_USER=todo_user
MYSQL_PASSWORD=todo_pass
API_PORT=3001
WEB_PORT=3000
NODE_ENV=development
```

Key Concepts

- Configure services without hardcoding credentials
- Compose reference: `${VAR:-default}` provides default if unset
- Inline override example:

```
MYSQL_ROOT_PASSWORD=secret MYSQL_DATABASE=test docker compose up --build
```

Docker Networking

Key Concepts

- Docker creates a bridge network per Compose project
- Containers communicate via service names (`db`, `api`, `web`)

- Omitting `ports:` keeps services internal
- Example: `web` has no ports → host cannot access `http://localhost:3000`
- `api` with `3001:3001` → host can access API

Database Persistence

Key Concepts

- Named volumes (`db_data`) persist MySQL data
- Reset database with:

```
docker compose down -v
```

- Without `-v`, data persists across container restarts

Service Exposure

Key Concepts

- `EXPOSE` → internal port hint only
- `ports:` → maps container port to host
- Keep frontend internal:

```
web:
    build: ./frontend
    # no ports mapping
```

Kubernetes Concepts

Example Deployment for API

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: todo-api
spec:
    replicas: 1
    selector:
        matchLabels:
            app: todo-api
    template:
        metadata:
            labels:
                app: todo-api
```

```
      spec:
        containers:
          - name: api
            image: todo-api:latest
            ports:
              - containerPort: 3001
            envFrom:
              - configMapRef:
                  name: todo-config
              - secretRef:
                  name: todo-secrets
```

Key Concepts

- `replicas` → number of pods
- `containerPort` → internal pod port
- `envFrom` → inject environment variables from ConfigMaps and Secrets
- Services control external exposure (similar to Docker `ports:`)

Best Practices

Key Concepts

- Use volumes for database persistence
- Use Docker networks to control service accessibility
- Keep frontend internal unless host access is needed
- Pass secrets via environment variables, not hardcoded
- Use Kubernetes for scaling and configuration management
- Reset database with `docker compose down -v` for a clean state