🚀 Full-Stack Tutorial: MySQL + Node/Express API + Express/EJS Frontend — Docker & Kubernetes

This tutorial walks you through **Part 1 (Docker)** and **Part 2 (Kubernetes)** for a simple full-stack project:

- **MySQL** database
- **Backend API**: Node.js + Express (generated with `npx express-generator`)
- **Frontend**: Express + EJS (generated with `npx express-generator --view=ejs`)
- All components run with **Docker Compose**
- Then deploy the same stack on **Kubernetes** via Docker Desktop

We'll build a tiny **Todo** app to keep things simple.

📁 Project Structure

```
fullstack-docker-k8s/
├── README.md
├── docker-compose.yml
├── .env                # shared defaults for docker compose (optional)
├── db/
│   └── init/           # optional: extra SQL files auto-run by MySQL
│       └── 00-init.sql
├── backend/            # created with: npx express-generator backend
│   ├── Dockerfile
│   ├── package.json
│   ├── app.js
│   ├── bin/www
│   ├── routes/
│   │   └── todos.js
│   ├── models/
│   │   ├── index.js
│   │   └── Todo.js
│   ├── utils/
│   │   └── wait-for-db.js
│   └── .env.example
└── frontend/           # created with: npx express-generator --view=ejs frontend
    ├── Dockerfile
    ├── package.json
    ├── app.js
    ├── bin/www
    ├── routes/
    │   └── index.js
    ├── views/
    │   ├── index.ejs
    │   └── error.ejs
    └── .env.example
```

What You'll Build (Simple Use Case)

- **Todos** stored in MySQL
- **API** exposes REST endpoints: GET/POST/PUT/DELETE /api/todos
- **Frontend** renders server-side HTML (EJS) and interacts with the API
- **Seed data** is automatically added at startup if the DB is empty

PART 1 — Docker Compose (Everything local)

1. `docker-compose.yml` (project root)

```
version: "3.9"

services:
    db:
        image: mysql:8.0
        container_name: todo-mysql
        environment:
            MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD:-rootpass}
            MYSQL_DATABASE: ${MYSQL_DATABASE:-todos}
            MYSQL_USER: ${MYSQL_USER:-todo_user}
            MYSQL_PASSWORD: ${MYSQL_PASSWORD:-todo_pass}
        ports:
            - "${MYSQL_PORT:-3306}:3306"
        volumes:
            - db_data:/var/lib/mysql
            - ./db/init:/docker-entrypoint-initdb.d
        healthcheck:
            test:
                [
                    "CMD-SHELL",
                    "mysqladmin ping -h localhost -p$${MYSQL_ROOT_PASSWORD} --
silent",
                ]
            interval: 5s
            timeout: 3s
            retries: 20
        networks:
            - appnet

    api:
        build: ./backend
        container_name: todo-api
        depends_on:
            db:
                condition: service_healthy
        environment:
            NODE_ENV: ${NODE_ENV:-development}
```

```
            PORT: ${API_PORT:-3001}
            DB_HOST: db
            DB_PORT: 3306
            DB_NAME: ${MYSQL_DATABASE:-todos}
            DB_USER: ${MYSQL_USER:-todo_user}
            DB_PASS: ${MYSQL_PASSWORD:-todo_pass}
        ports:
            - "${API_PORT:-3001}:3001"
        networks:
            - appnet

    web:
        build: ./frontend
        container_name: todo-web
        depends_on:
            - api
        environment:
            NODE_ENV: ${NODE_ENV:-development}
            PORT: ${WEB_PORT:-3000}
            API_BASE_URL: "http://api:3001"
        ports:
            - "${WEB_PORT:-3000}:3000"
        networks:
            - appnet

volumes:
    db_data:

networks:
    appnet:
        driver: bridge
```

> Save this as `docker-compose.yml` in the project root.

2. Optional `.env` (project root)

```
MYSQL_ROOT_PASSWORD=rootpass
MYSQL_DATABASE=todos
MYSQL_USER=todo_user
MYSQL_PASSWORD=todo_pass

API_PORT=3001
WEB_PORT=3000
```

Place this file at project root if you want environment defaults picked up by `docker compose`.

---

3. Database init script (optional)

`db/init/00-init.sql` — can be empty or contain SQL to initialize database. MySQL image runs anything in `/docker-entrypoint-initdb.d` only when initializing a fresh DB.

Example (optional):

```
-- db/init/00-init.sql
-- runs only on first container start (when DB is empty)
-- we don't need to insert todos here because backend will seed automatically
```

---

4. Backend (API) — scaffold + files

Scaffold

```
# from project root
npx express-generator backend
cd backend
npm install
npm i sequelize mysql2 dotenv
```

> `express-generator` creates the baseline `app.js`, `bin/www`, `routes`, `views` (we won't use views in API), and `package.json`.

`backend/.env.example`

```
NODE_ENV=development
PORT=3001

DB_HOST=localhost
DB_PORT=3306
DB_NAME=todos
DB_USER=todo_user
DB_PASS=todo_pass
```

`backend/Dockerfile`

```
# backend/Dockerfile
FROM node:20-alpine

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm ci --omit=dev

COPY . .
```

```
ENV PORT=3001
EXPOSE 3001

# Wait for DB to be reachable then start the app
CMD ["sh", "-c", "node ./utils/wait-for-db.js && node ./bin/www"]
```

backend/utils/wait-for-db.js

```
// backend/utils/wait-for-db.js
// A tiny script that attempts to connect to MySQL before letting the container
start the API

const mysql = require("mysql2/promise");

const {
    DB_HOST = "localhost",
    DB_PORT = "3306",
    DB_USER = "root",
    DB_PASS = "",
} = process.env;

const sleep = (ms) => new Promise((r) => setTimeout(r, ms));

(async () => {
    const max = 30; // attempts
    for (let i = 1; i <= max; i++) {
        try {
            const conn = await mysql.createConnection({
                host: DB_HOST,
                port: DB_PORT,
                user: DB_USER,
                password: DB_PASS,
            });
            await conn.ping();
            await conn.end();
            console.log("✅ Database reachable");
            process.exit(0);
        } catch (e) {
            console.log(`⏳ Waiting for DB... (${i}/${max})`);
            await sleep(2000);
        }
    }
    console.error("❌ Could not connect to DB in time.");
    process.exit(1);
})();
```

backend/models/Todo.js

```
// backend/models/Todo.js
const { DataTypes } = require("sequelize");

module.exports = (sequelize) => {
    const Todo = sequelize.define(
        "Todo",
        {
            id: {
                type: DataTypes.INTEGER,
                primaryKey: true,
                autoIncrement: true,
            },
            title: { type: DataTypes.STRING(255), allowNull: false },
            completed: { type: DataTypes.BOOLEAN, defaultValue: false },
        },
        {
            tableName: "todos",
            timestamps: true,
        }
    );
    return Todo;
};
```

backend/models/index.js

```
// backend/models/index.js
const { Sequelize } = require("sequelize");
require("dotenv").config();

const {
    DB_HOST = "localhost",
    DB_PORT = "3306",
    DB_NAME = "todos",
    DB_USER = "root",
    DB_PASS = "",
    NODE_ENV = "development",
} = process.env;

const sequelize = new Sequelize(DB_NAME, DB_USER, DB_PASS, {
    host: DB_HOST,
    port: DB_PORT,
    dialect: "mysql",
    logging: NODE_ENV === "development" ? console.log : false,
});

const Todo = require("./Todo")(sequelize);

async function initAndSeed() {
    await sequelize.authenticate();
    // create / migrate tables
```

```
    await sequelize.sync({ alter: true });

    const count = await Todo.count();
    if (count === 0) {
        await Todo.bulkCreate([
            { title: "Learn Docker", completed: false },
            { title: "Wire up API", completed: false },
            { title: "Build EJS frontend", completed: false },
        ]);
        console.log("🌱 Seeded initial todos");
    }
}

module.exports = { sequelize, Todo, initAndSeed };
```

backend/routes/todos.js

```
// backend/routes/todos.js
const express = require("express");
const router = express.Router();
const { Todo } = require("../models");

// GET /api/todos
router.get("/", async (req, res, next) => {
    try {
        const todos = await Todo.findAll({ order: [["id", "ASC"]] });
        res.json(todos);
    } catch (e) {
        next(e);
    }
});

// POST /api/todos
router.post("/", async (req, res, next) => {
    try {
        const { title } = req.body;
        const todo = await Todo.create({ title, completed: false });
        res.status(201).json(todo);
    } catch (e) {
        next(e);
    }
});

// PUT /api/todos/:id
router.put("/:id", async (req, res, next) => {
    try {
        const { title, completed } = req.body;
        const todo = await Todo.findByPk(req.params.id);
        if (!todo) return res.status(404).json({ error: "Not found" });
        await todo.update({ title, completed });
        res.json(todo);
```

```
    } catch (e) {
        next(e);
    }
});

// DELETE /api/todos/:id
router.delete("/:id", async (req, res, next) => {
    try {
        const todo = await Todo.findByPk(req.params.id);
        if (!todo) return res.status(404).json({ error: "Not found" });
        await todo.destroy();
        res.status(204).end();
    } catch (e) {
        next(e);
    }
});

module.exports = router;
```

`backend/app.js` (patch the generated app)

```
// backend/app.js
const createError = require("http-errors");
const express = require("express");
const path = require("path");
const cookieParser = require("cookie-parser");
const logger = require("morgan");

const todosRouter = require("./routes/todos");
const { initAndSeed } = require("./models");

const app = express();

app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));

// API routes
app.use("/api/todos", todosRouter);

// health check
app.get("/healthz", (_req, res) => res.json({ ok: true }));

// catch 404
app.use(function (req, res, next) {
    next(createError(404));
});

// error handler
```

```
app.use(function (err, req, res, _next) {
    res.status(err.status || 500).json({ error: err.message });
});

// initialize DB and seed data
initAndSeed().catch((err) => {
    console.error("DB init failed:", err);
    process.exit(1);
});

module.exports = app;
```

> The `bin/www` generated by express-generator can remain. It reads `process.env.PORT` and starts the app.

---

5. Frontend (Express + EJS) — scaffold + files

Scaffold

```
# from project root
npx express-generator --view=ejs frontend
cd frontend
npm install
npm i dotenv
```

`frontend/.env.example`

```
NODE_ENV=development
PORT=3000
API_BASE_URL=http://localhost:3001
```

`frontend/Dockerfile`

```
# 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"]
```

`frontend/app.js` (patch)

```
// frontend/app.js
const createError = require("http-errors");
const express = require("express");
const path = require("path");
const cookieParser = require("cookie-parser");
const logger = require("morgan");
require("dotenv").config();

const indexRouter = require("./routes/index");

const app = express();

// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");

app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));

app.use("/", indexRouter);

app.get("/healthz", (_req, res) => res.json({ ok: true }));

// catch 404 and error handler
app.use(function (req, res, next) {
    next(createError(404));
});

app.use(function (err, req, res, _next) {
    res.status(err.status || 500);
    res.render("error", { message: err.message, error: err });
});

module.exports = app;
```

`frontend/routes/index.js`

```
// frontend/routes/index.js
const express = require("express");
const router = express.Router();
```

```
const API_BASE_URL = process.env.API_BASE_URL || "http://localhost:3001";

// helper to call API using fetch
async function callApi(path, options = {}) {
    const url = `${API_BASE_URL}${path}`;
    const res = await fetch(url, {
        headers: { "Content-Type": "application/json" },
        ...options,
    });
    if (!res.ok) {
        const errText = await res.text();
        throw new Error(`API Error: ${res.status} ${errText}`);
    }
    if (res.status !== 204) {
        return res.json();
    }
    return null;
}

// GET /
router.get("/", async (req, res, next) => {
    try {
        const todos = await callApi("/api/todos");
        res.render("index", { title: "Todos", todos });
    } catch (e) {
        next(e);
    }
});

// POST /add
router.post("/add", async (req, res, next) => {
    try {
        const title = req.body.title;
        if (title?.trim()) {
            await callApi("/api/todos", {
                method: "POST",
                body: JSON.stringify({ title }),
            });
        }
        res.redirect("/");
    } catch (e) {
        next(e);
    }
});

// POST /toggle/:id
router.post("/toggle/:id", async (req, res, next) => {
    try {
        const id = req.params.id;
        const { title, completed } = req.body;
        await callApi(`/api/todos/${id}`, {
            method: "PUT",
            body: JSON.stringify({ title, completed: completed === "true" }),
        });
```

```
        res.redirect("/");
    } catch (e) {
        next(e);
    }
});

// POST /delete/:id
router.post("/delete/:id", async (req, res, next) => {
    try {
        await callApi(`/api/todos/${req.params.id}`, { method: "DELETE" });
        res.redirect("/");
    } catch (e) {
        next(e);
    }
});

module.exports = router;
```

frontend/views/index.ejs

```
<!DOCTYPE html>
<html>
    <head>
        <title><% title %></title>
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="stylesheet" href="/stylesheets/style.css" />
    </head>
    <body>
        <div class="container">
            <h1>Todos</h1>
            <form action="/add" method="post" style="margin-bottom: 1rem">
                <input
                    type="text"
                    name="title"
                    placeholder="New todo..."
                    required
                />
                <button type="submit">Add</button>
            </form>

            <ul>
                <% for (const t of todos) { %>
                <li style="margin-bottom: 0.5rem">
                    <form
                        action="/toggle/<%= t.id %>"
                        method="post"
                        style="display: inline"
                    >
                        <input
                            type="hidden"
                            name="title"
```

```
                            value="<%= t.title %>"
                        />
                        <input
                            type="hidden"
                            name="completed"
                            value="<%= !t.completed %>"
                        />
                        <button type="submit" style="margin-right: 0.5rem">
                            <%= t.completed ? '☑' : '☐' %>
                        </button>
                    </form>
                    <strong><%= t.title %></strong>
                    <form
                        action="/delete/<%= t.id %>"
                        method="post"
                        style="display: inline; margin-left: 0.5rem"
                    >
                        <button type="submit">🗑</button>
                    </form>
                </li>
                <% } %>
            </ul>
        </div>
    </body>
</html>
```

---

6. Build & Run with Docker Compose

From project root:

```
# Build images
docker compose build

# Start the stack (foreground)
docker compose up

# Or run detached
docker compose up -d

# Check logs for api
docker compose logs -f api

# Delete persisted database data
docker compose down -v
# The -v flag removes all named volumes declared in the compose file, including
db_data
```

- Frontend: `http://localhost:3000`
- API: `http://localhost:3001/api/todos`

- MySQL: `localhost:3306` (use credentials from .env)

Notes

- `depends_on` with `condition: service_healthy` ensures API waits for MySQL healthcheck.
- The backend's `wait-for-db.js` adds an extra safety: try connecting before starting the Node process.

PART 2 — Kubernetes (Docker Desktop)

> This section shows how to run same stack on Kubernetes (local cluster via Docker Desktop). If you prefer, use minikube / kind — the manifest files are portable.

1. Enable Kubernetes in Docker Desktop

1. Open Docker Desktop → Settings → Kubernetes.
2. Check **Enable Kubernetes**. Apply & Restart.
3. Verify cluster:

```
kubectl version --client
kubectl get nodes
```

You should see at least one node (Docker Desktop).

2. Image strategy

Two options:

1. **Push images to a registry** (recommended and portable).

   ```
   # From project root after docker compose build (or docker build)
   docker tag fullstack-docker-k8s-backend:latest yourdockerhubuser/todo-
   api:1.0.0
   docker push yourdockerhubuser/todo-api:1.0.0

   docker tag fullstack-docker-k8s-frontend:latest yourdockerhubuser/todo-
   web:1.0.0
   docker push yourdockerhubuser/todo-web:1.0.0
   ```

2. **Use local images**: Docker Desktop's Kubernetes can use local images built by your Docker daemon (no push). If not, push to local registry or use `kind load docker-image` for kind.

3. Kubernetes manifests (create `k8s/` folder)

Below are example manifests. Save them to files under `k8s/`.

`k8s/namespace.yaml`

```
apiVersion: v1
kind: Namespace
metadata:
    name: todos
```

k8s/secrets.yaml

> Store sensitive values in Secrets (base64 encoded). Replace base64 values with your own if you change passwords.

```
apiVersion: v1
kind: Secret
metadata:
    name: db-secret
    namespace: todos
type: Opaque
data:
    MYSQL_ROOT_PASSWORD: cm9vdHBhc3M= # "rootpass"
    MYSQL_PASSWORD: dG9kb19wYXNz # "todo_pass"
```

You can generate base64:

```
echo -n 'rootpass' | base64
echo -n 'todo_pass' | base64
```

k8s/configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: app-config
    namespace: todos
data:
    MYSQL_DATABASE: todos
    MYSQL_USER: todo_user
    DB_HOST: mysql
    DB_PORT: "3306"
    DB_NAME: todos
    DB_USER: todo_user
    API_PORT: "3001"
    WEB_PORT: "3000"
    API_BASE_URL: "http://api:3001"
    NODE_ENV: "production"
```

`k8s/mysql-pvc.yaml`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: mysql-pvc
    namespace: todos
spec:
    accessModes:
        - ReadWriteOnce
    resources:
        requests:
            storage: 2Gi
```

`k8s/mysql-deployment.yaml` (deploy + service)

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: mysql
    namespace: todos
spec:
    replicas: 1
    selector:
        matchLabels:
            app: mysql
    template:
        metadata:
            labels:
                app: mysql
        spec:
            containers:
                - name: mysql
                  image: mysql:8.0
                  ports:
                      - containerPort: 3306
                  env:
                      - name: MYSQL_ROOT_PASSWORD
                        valueFrom:
                            secretKeyRef:
                                name: db-secret
                                key: MYSQL_ROOT_PASSWORD
                      - name: MYSQL_DATABASE
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: MYSQL_DATABASE
                      - name: MYSQL_USER
                        valueFrom:
                            configMapKeyRef:
```

```
                                name: app-config
                                key: MYSQL_USER
                        - name: MYSQL_PASSWORD
                          valueFrom:
                              secretKeyRef:
                                  name: db-secret
                                  key: MYSQL_PASSWORD
                    volumeMounts:
                        - name: mysql-data
                          mountPath: /var/lib/mysql
                    readinessProbe:
                        exec:
                            command:
                                - bash
                                - -c
                                - mysqladmin ping -h 127.0.0.1 -prootpass || exit 1
                        initialDelaySeconds: 10
                        periodSeconds: 5
            volumes:
                - name: mysql-data
                  persistentVolumeClaim:
                      claimName: mysql-pvc
---
apiVersion: v1
kind: Service
metadata:
    name: mysql
    namespace: todos
spec:
    selector:
        app: mysql
    ports:
        - port: 3306
          targetPort: 3306
    type: ClusterIP
```

> **Important:** If you change the root pass in Secret, adjust the readiness probe command or use an env var.

k8s/api-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: api
    namespace: todos
spec:
    replicas: 1
    selector:
        matchLabels:
            app: api
```

```
    template:
        metadata:
            labels:
                app: api
        spec:
            containers:
                - name: api
                  image: yourdockerhubuser/todo-api:1.0.0
                  ports:
                      - containerPort: 3001
                  env:
                      - name: NODE_ENV
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: NODE_ENV
                      - name: PORT
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: API_PORT
                      - name: DB_HOST
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: DB_HOST
                      - name: DB_PORT
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: DB_PORT
                      - name: DB_NAME
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: DB_NAME
                      - name: DB_USER
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: DB_USER
                      - name: DB_PASS
                        valueFrom:
                            secretKeyRef:
                                name: db-secret
                                key: MYSQL_PASSWORD
                  readinessProbe:
                      httpGet:
                          path: /healthz
                          port: 3001
                      initialDelaySeconds: 5
                      periodSeconds: 5
---
apiVersion: v1
```

```
kind: Service
metadata:
    name: api
    namespace: todos
spec:
    selector:
        app: api
    ports:
        - port: 3001
          targetPort: 3001
    type: ClusterIP
```

k8s/web-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: web
    namespace: todos
spec:
    replicas: 1
    selector:
        matchLabels:
            app: web
    template:
        metadata:
            labels:
                app: web
        spec:
            containers:
                - name: web
                  image: yourdockerhubuser/todo-web:1.0.0
                  ports:
                      - containerPort: 3000
                  env:
                      - name: NODE_ENV
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: NODE_ENV
                      - name: PORT
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: WEB_PORT
                      - name: API_BASE_URL
                        valueFrom:
                            configMapKeyRef:
                                name: app-config
                                key: API_BASE_URL
                  readinessProbe:
```

```
                    httpGet:
                        path: /healthz
                        port: 3000
                    initialDelaySeconds: 5
                    periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
    name: web
    namespace: todos
spec:
    selector:
        app: web
    ports:
        - port: 3000
          targetPort: 3000
    type: NodePort
```

(Optional) `k8s/ingress.yaml` — use if you have an Ingress controller

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
    name: web-ingress
    namespace: todos
    annotations:
        kubernetes.io/ingress.class: nginx
spec:
    rules:
        - host: todos.localtest.me
          http:
              paths:
                  - path: /
                    pathType: Prefix
                    backend:
                        service:
                            name: web
                            port:
                                number: 3000
```

> `localtest.me` resolves to `127.0.0.1` (handy for local testing).

4. Apply manifests to the cluster

```
kubectl apply -f k8s/namespace.yaml
kubectl apply -f k8s/secrets.yaml
kubectl apply -f k8s/configmap.yaml
kubectl apply -f k8s/mysql-pvc.yaml
```

```
kubectl apply -f k8s/mysql-deployment.yaml
kubectl apply -f k8s/api-deployment.yaml
kubectl apply -f k8s/web-deployment.yaml
# optional ingress
kubectl apply -f k8s/ingress.yaml
```

Check resources:

```
kubectl -n todos get pods,svc,deploy,pvc
kubectl -n todos logs deploy/api
kubectl -n todos logs deploy/web
```

Accessing the frontend

- If using **NodePort**, get port:

  ```
  kubectl -n todos get svc web -o yaml
  ```

  or

  ```
  kubectl -n todos get svc web
  # open http://localhost:<nodePort>
  ```

- If using **port-forward**:

  ```
  kubectl -n todos port-forward svc/web 3000:3000
  # open http://localhost:3000
  ```

- If using **Ingress** with `todos.localtest.me`, add host / access at `http://todos.localtest.me/`.

Why Kubernetes vs Docker Compose (short)

- **Docker Compose** is excellent for quick local development and testing of multi-container apps.
- **Kubernetes** is declarative, built for production-scale orchestration: self-healing, rolling updates, secrets/configmaps, persistent volume management, service discovery and load balancing.
- With Kubernetes you `kubectl apply` manifests; the control plane manages scheduling, scaling, and service networking.

Environment variables summary

Component	Env var	Purpose
MySQL	`MYSQL_*`	DB root/user/password and initial database
API	`DB_HOST`	points to `db` (docker) or `mysql` (k8s service)
API	`DB_NAME`	DB name
API	`DB_USER`	DB user
API	`DB_PASS`	DB password
API	`PORT`	API port (default 3001)
Frontend	`API_BASE_URL`	Where web calls the API
Frontend	`PORT`	Frontend port (default 3000)

Useful commands

Docker (compose)

```
# build
docker compose build

# up
docker compose up -d

# view logs
docker compose logs -f api
docker compose logs -f web

# stop and remove
docker compose down -v
```

Kubernetes

```
# apply all manifests in k8s/
kubectl apply -f k8s/

# watch pods
kubectl -n todos get pods -w

# inspect logs
kubectl -n todos logs deploy/api
kubectl -n todos logs -f deploy/web

# port-forward to frontend locally
kubectl -n todos port-forward svc/web 3000:3000
```

```
# teardown
kubectl -n todos delete namespace todos
```

Troubleshooting tips

- **DB connection refused**: check MySQL container logs; ensure `.env` credentials match those used by the API; ensure healthcheck passed.
- **Sequelize sync errors**: check DB user privileges; try connecting with `mysql -u user -ppass -h host` from inside a container.
- **Frontend can't reach API in Docker**: ensure frontend uses `http://api:3001` inside compose (service name), and use `API_BASE_URL` env variable for runtime override.
- **K8s image not found**: either push image to a registry or use Docker Desktop's shared daemon for local images; confirm `image` name in manifests matches what you built/pushed.

Final notes & checklist

- Both apps were scaffolded with `npx express-generator` (backend plain, frontend with `--view=ejs`).
- Backend uses **Sequelize** and **mysql2** to talk to MySQL and seeds initial data when the DB is empty.
- Everything is configured to accept environment variables so it can run both in Docker Compose and Kubernetes.
- You can copy-paste this entire file into `README.md` (or another `.md` file) — it's formatted as full Markdown and contains all code blocks.

Happy building! If you'd like, I can:

- generate `package.json` examples for each app,
- produce `Dockerfile` optimizations (multi-stage build),
- or convert the k8s manifests into a single `kustomization.yaml` or Helm chart.