# Introduzione al deployment (guida passo–passo)

## Obiettivo

- Pubblicare un backend Express su un PaaS (Render o Railway; nota su Vercel in fondo).
- Collegarlo a **MongoDB Atlas**.
- Impostare **variabili d’ambiente** in modo sicuro.
- Testare l’API online con browser/Postman.

Nota: i passaggi sono **molto concreti**. Gli studenti possono seguirli uno ad uno.

## Prima di iniziare: “check di produzione” in locale

1. **package.json**: serve lo script di start

```
{
  "name": "my-api",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",         // ← usato in produzione
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4",
    "mongoose": "^8",
    "cors": "^2",
    "dotenv": "^16"
  }
}
```

2. **server.js**: usare la porta del provider

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');

const app = express();
app.use(cors());          // in prod, restringi il CORS se vuoi
app.use(express.json());

app.get('/api/health', (req, res) => res.json({ status: 'ok' }));

const PORT = process.env.PORT || 3000;    // ← fondamentale
app.listen(PORT, () => console.log(`Server on http://localhost:${PORT}`));
```

3. **Connessione Mongo** (esempio):

```
const mongoose = require('mongoose');
mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('MongoDB connesso'))
  .catch(err => { console.error('Errore MongoDB:', err.message);
process.exit(1); });
```

4. **.env** (solo in locale; NON committare)

```
PORT=3000
MONGO_URI=mongodb+srv://<user>:<pass>@<cluster>.mongodb.net/mydb
JWT_SECRET=<valore-segreto>
NODE_ENV=development
```

5. **.gitignore**: non mandare su Git cose sensibili/voluminose

```
node_modules/
.env
uploads/
```

6. Avvia **in locale**:

```
npm install
npm run dev
# prova: http://localhost:3000/api/health
```

Se tutto ok in locale, si passa al cloud.

# Preparare MongoDB Atlas (una volta sola)

1. Vai su **cloud.mongodb.com**, crea un account gratuito.
2. Crea un **Cluster** (free tier).
3. Crea un **Database User** (username/password) → con permessi “Read and write to any database” o su un db specifico.
4. In **Network Access**, consenti il tuo IP o (per test) `0.0.0.0/0` (tutti).

   Attenzione: `0.0.0.0/0` è comodo per laboratorio, **non** per produzione.

5. Recupera la **connection string** (esempio):

   ```
   mongodb+srv://<USER>:<PASS>@<CLUSTER>.mongodb.net/mydb?
   retryWrites=true&w=majority
   ```

6. Questa stringa **NON** va nel codice: verrà messa tra le variabili d’ambiente del provider (Render/Railway).

# Opzione A: Deploy su Render (molto semplice)

1. **Repo su GitHub**
    - `git init` → `git add .` → `git commit -m "init"`
    - crea un repo su GitHub e fai `git remote add origin ...` + `git push -u origin main`
2. Su **render.com** → "New" → **Web Service**
    - Connetti il tuo **repository GitHub**
    - "Root Directory": la root del progetto (se usi `/src`, indica correttamente gli entrypoint)
    - "Build Command": spesso **vuoto** per Node puro (Render fa `npm install` di default)
    - "Start Command": **`npm start`** (usa quello del package.json)
3. **Environment**
    - Imposta le **Environment Variables** (Settings → Environment):
        - `MONGO_URI` = (stringa Atlas)
        - `JWT_SECRET` = (chiave segreta)
        - **Non** impostare `PORT`: Render assegna automaticamente `PORT` al container.
4. **Deploy**
    - Render fa il build e avvia.
    - Se tutto ok, ti mostra l'URL pubblico, tipo: `https://my-api.onrender.com`
5. **Test**
    - Apri il browser: `https://my-api.onrender.com/api/health`
    - Postman: prova le rotte CRUD.

⚠ Se vedi errori:

- controlla i **Logs** su Render
- verifica che `MONGO_URI` sia corretta e che l'utente DB esista
- se time-out su Mongo, controlla **Network Access** su Atlas (IP consentito)
- se il server non parte, verifica `npm start` e che **PORT** venga letta da `process.env.PORT`

# Opzione B: Deploy su Railway

1. Repo su GitHub pronto (come sopra).
2. Vai su **railway.app** → New Project → **Deploy from GitHub repo**.
3. Seleziona il repo.
4. Imposta **Variables** (sezione Variables):
    - `MONGO_URI`, `JWT_SECRET`
5. Automaticamente Railway imposta la porta in env; il tuo server deve usare `process.env.PORT`.
6. Controlla i **Logs** durante il build/avvio.
7. Testa con l'URL pubblico: `https://<project>.up.railway.app/api/health`

Railway ha anche plugin/database integrati. Per Mongo è comodo restare su Atlas.

# Nota su Vercel (meglio per frontend)

Vercel è perfetto per frontend (Next.js) e funzioni serverless; un'app Express "classica" richiede un adattamento (serverless functions o `vercel.json` specifico).
Per una prima esperienza **senza frizioni**, consiglio Render o Railway per il backend Express.

# Test dell'API online (curl/Postman)

Con l'URL del tuo servizio (es. `https://my-api.onrender.com`):

**Health check**

```
curl https://my-api.onrender.com/api/health
# → {"status":"ok"}
```

**Esempio CRUD (se hai /api/notes)**

```
# CREATE
curl -X POST https://my-api.onrender.com/api/notes \
  -H "Content-Type: application/json" \
  -d '{"title":"Prima nota","content":"Contenuto"}'

# LIST
curl https://my-api.onrender.com/api/notes

# READ ONE
curl https://my-api.onrender.com/api/notes/<ID>

# UPDATE
curl -X PATCH https://my-api.onrender.com/api/notes/<ID> \
  -H "Content-Type: application/json" \
  -d '{"title":"Titolo aggiornato"}'
```

```
# DELETE
curl -X DELETE https://my-api.onrender.com/api/notes/<ID>
```

Se usi **JWT**:

- fai login → ricevi `token`
- aggiungi header: `Authorization: Bearer <token>`

# Variabili d’ambiente: cosa mettere online

Nel pannello del provider (Render/Railway), configura:

- `MONGO_URI` → stringa Atlas
- `JWT_SECRET` → stringa random lunga (es. 32+ char)
- Se vuoi: `NODE_ENV=production`

> **Mai** mettere queste cose nel repo Git.
> Se cambiano, aggiorni dal pannello e redeployi.

# Differenze tra dev e production

- **Dev**
  - log verbosi (`console.log`, stack errori)
  - CORS permissivo (localhost varie porte)
  - reload automatico con `nodemon`
- **Prod**
  - log strutturati (winston), niente stack in risposta
  - CORS mirato (solo il tuo frontend)
  - rate limiting + helmet
  - gestione errori coerente e “sobria”
  - nessun `nodemon` (usa `node server.js`)

Sulla piattaforma puoi aggiungere:

- `helmet` per header di sicurezza
- `express-rate-limit` per limitare richieste su endpoint sensibili
- CORS con **whitelist** (soli domini autorizzati)

## Checklist “pronto al deploy”

- `npm start` avvia il server senza errori
- `process.env.PORT` usato correttamente (niente hard–code della porta)
- `dotenv` usato SOLO in locale (su cloud le env arrivano dal pannello)
- `MONGO_URI` di Atlas funzionante (utente/password, Network Access ok)
- `errorHandler` presente, niente stack trace in risposta
- `CORS` configurato (almeno permissivo per test; poi restringi)
- `.gitignore` con `.env`, `node_modules`, `uploads`
- Health check `/api/health` risponde

## Troubleshooting rapido

- **App non parte** → guarda i Logs; spesso è `PORT` non letta o `MONGO_URI` errata.
- **Errore Mongo “authentication failed”** → controlla user/password e **DB name** nella stringa.
- **Timeout Mongo** → IP non autorizzato in Atlas (aggiungi IP del provider o `0.0.0.0/0` per test).
- **CORS error sul browser** → configura `cors()` con `origin` corretto (dominio del tuo frontend).
- **404 su tutte le rotte** → path base errato (`/api/...`) o middleware ordine sbagliato.
- **“App crashed” su Render/Railway** → controlla `Start Command` (deve essere `npm start`), logs e versioni Node.