Express Microservice Project Structure

Container-ready microservice with health checks, graceful shutdown, and Docker support.

#express #nodejs #microservice #docker #container

PNG PDF Copy Prompt

Project Directory

```
myproject/
  src/
    index.js  Server startup …
    app.js  Express app fac…
    config/
      index.js  Environment-bas…
    routes/
      index.js
      health.js  /health and /re…
    middleware/
      request-id.js  Trace ID for re…
      logger.js  Structured JSON…
      error-handler.js
    services/
      index.js
    utils/
      logger.js  Pino or Winston…
      shutdown.js  Graceful shutdo…
  tests/
    health.test.js
    integration/
  Dockerfile  Multi-stage bui…
  docker-compose.yml  Local developme…
  .dockerignore
  package.json
  .env
  .env.example
  .gitignore
```

Why This Structure?

Built for containers from the start. Health endpoints let orchestrators know service state. Graceful shutdown prevents dropped connections during deploys. Structured logging makes debugging in distributed systems possible.

Key Directories

routes/health.js - `/health` for liveness, `/ready` for readiness probes

utils/shutdown.js - SIGTERM handler, connection draining

middleware/request-id.js - Correlation ID for distributed tracing

middleware/logger.js - JSON logs with request context

Graceful Shutdown

```
// src/utils/shutdown.js
function gracefulShutdown(server) {
  process.on('SIGTERM', () => {
    console.log('SIGTERM received, shutting down...');
    server.close(() => {
      console.log('Server closed');
      process.exit(0);
    });
  });
}
```

Health Endpoints

```
// src/routes/health.js
router.get('/health', (req, res) => {
  res.json({ status: 'ok' });
});

router.get('/ready', async (req, res) => {
  const dbOk = await checkDatabase();
  res.status(dbOk ? 200 : 503).json({ ready: dbOk });
});
```

Getting Started

1. `npm init -y`
2. `npm install express dotenv pino`
3. `cp .env.example .env`
4. `docker-compose up` # Start with Docker
5. `npm run dev` # Or run directly

When To Use This

- Deploying to Kubernetes or Docker Swarm
- Building a system of small, focused services
- Need zero-downtime deployments
- Distributed systems with observability needs
- Services behind a load balancer

Trade-offs

More moving parts - Docker, health checks, logging all add complexity

Operational overhead - Need container orchestration knowledge

Not for monoliths - Overkill if building a single app

Best Practices

- Use multi-stage Docker builds for smaller images
- Log in JSON format for log aggregators
- Set appropriate timeouts for graceful shutdown
- Include build info in `/health` response
- Use `pino` for high-performance logging