Perfect 👍 Let's continue with **Topic 21: Deployment & DevOps in Node.js** in the same structured way.

21. Deployment & DevOps in Node.js

Deploying a Node.js application involves packaging, configuring environments, monitoring, and ensuring scalability & reliability. DevOps practices help automate these processes.

---

1. Preparing a Node.js App for Deployment

- **Environment Variables**
  - Store secrets (DB URL, API keys) in `.env` (use `dotenv` package).
- **Build Process**
  - Transpile (if using TypeScript/Babel).
  - Bundle frontend (if full-stack project).
- **Production Settings**
  - Disable debug logs.
  - Enable compression & caching.
  - Use a process manager (PM2, Forever).

---

2. Deployment Options

a) Traditional Hosting

- Run on **VPS/Server** (Linux, Windows).
- Use **NGINX/Apache** as reverse proxy.
- Example: DigitalOcean, Linode.

b) Cloud Platforms

- **AWS EC2, Elastic Beanstalk**
- **Google Cloud Run, App Engine**
- **Azure App Service**
- Autoscaling, monitoring, load balancing built-in.

c) Containerization

- **Dockerize Node.js app**

```
FROM node:18
WORKDIR /app
```

```
COPY package*.json ./
RUN npm install --production
COPY . .
CMD ["node", "server.js"]
```

- Deploy with **Kubernetes (K8s)** for scaling.

d) Serverless

- Deploy functions instead of full servers.
- AWS Lambda, Google Cloud Functions, Vercel, Netlify.
- Great for event-driven APIs.

---

3. Process Management

- **PM2** (most popular for Node.js)

```
npm install -g pm2
pm2 start app.js -i max   # cluster mode
pm2 logs
pm2 restart all
```

- Handles:
  - Auto-restarts on crash.
  - Load balancing (cluster mode).
  - Monitoring & logs.
- **Forever**: simpler alternative.

---

4. Reverse Proxy Setup (NGINX)

- Improves performance & handles SSL/TLS.
- Example config:

```
server {
    listen 80;
    server_name example.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
```

```
    }
}
```

---

5. CI/CD (Continuous Integration & Deployment)

- **Tools**: GitHub Actions, GitLab CI, Jenkins, CircleCI.
- Workflow:
    1. Push code → Trigger build pipeline.
    2. Run tests → Build Docker image.
    3. Deploy to staging/production.

Example: GitHub Actions (`.github/workflows/deploy.yml`)

```yaml
name: Deploy Node.js App
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: 18
      - run: npm install
      - run: npm test
```

---

6. Monitoring & Logging

- **PM2 Monitoring** → `pm2 monit`
- **Tools**:
    - Log management: Winston, Morgan, ELK stack.
    - Monitoring: Prometheus + Grafana, New Relic, Datadog.
    - Error tracking: Sentry.

---

7. Scaling Node.js Apps

- **Vertical Scaling**: Increase CPU/RAM of server.
- **Horizontal Scaling**: Run multiple app instances behind load balancer.
- **Clustering**: Node.js cluster module or PM2 `-i max`.
- **Kubernetes**: Auto-scale containers.

---

8. Best Practices

- Use `.env` & **12-factor app principles**.
- Deploy with **Docker** for portability.
- Keep logs centralized (Winston, ELK).
- Always run Node.js in **production mode** (`NODE_ENV=production`).
- Automate deployments with CI/CD.
- Monitor health & auto-restart on crash (PM2, Docker, K8s).
- Secure with **NGINX reverse proxy + SSL/TLS**.

☑ That completes **Deployment & DevOps in Node.js**.
Would you like me to now cover **Topic 22: Advanced Patterns & Best Practices** (final core technical section before wrapping up)?