NestJS + Angular on AWS Kubernetes

Complete Guide: Docker, Kubernetes, RDS PostgreSQL, S3 Storage

Production-Ready Microservices Architecture

Publisher: Bithost

sales@bithost.in

www.bithost.in

Document Type: Complete Technical Implementation Guide (15 Sections)

Reading Time: ~2 Hour

Version: 1.0 | **Published:** Jan 26, 2026

Stack: NestJS 10+, Angular 17+, Docker, Kubernetes, AWS EKS, PostgreSQL 15, S3

📚 Complete Table of Contents (15 SECTIONS)

1. Executive Summary & Architecture Vision

This guide provides a complete blueprint for deploying a modern full-stack application (NestJS API + Angular Frontend) on AWS Kubernetes (EKS) with containerization via Docker, persistent data in RDS PostgreSQL, and file storage in S3.

Architecture Overview

Three-Layer Architecture:

- **Presentation Layer:** Angular SPA served via S3 + CloudFront CDN
- **Application Layer:** NestJS API running in EKS pods
- **Data Layer:** RDS PostgreSQL + S3 file storage

Key Features

Component	Technology	Purpose
API Server	NestJS 10+	Scalable backend with dependency injection
Frontend	Angular 17+	Responsive SPA with RxJS
Containerization	Docker	Consistent deployment across environments
Orchestration	Kubernetes (EKS)	Auto-scaling, self-healing, rolling updates
Database	RDS PostgreSQL	Managed relational database with backups
File Storage	AWS S3	Scalable object storage for media

Benefits of This Approach

- ✅ **High Availability:** Multi-AZ deployment with auto-failover
- ✅ **Auto-Scaling:** Horizontal pod autoscaling based on metrics
- ✅ **Zero Downtime:** Rolling updates without service interruption
- ✅ **Separation of Concerns:** Frontend, API, and data independently scaled
- ✅ **Cost Efficient:** Pay-per-use with auto-scaling
- ✅ **Easy CI/CD:** Git-driven deployments with ArgoCD
- ✅ **Production Grade:** Built-in monitoring and logging

2. Infrastructure Planning & AWS Selection

AWS Services Selection

Service	Configuration	Rationale
EKS	3-5 worker nodes (t3.medium-t3.large)	Managed Kubernetes, AWS integration
RDS PostgreSQL	db.t3.large Multi-AZ	Automated backups, replication, patches
S3	Standard + Intelligent-Tiering	Durable, scalable file storage
ECR	Private container registry	Store Docker images securely
CloudFront	CDN with S3 origin	Serve Angular app from edge

Service	Configuration	Rationale
CloudWatch	Logs + Metrics + Alarms	Comprehensive observability

Cost Estimation (Monthly)

Component	Low Usage	Medium Usage	High Usage
EKS Cluster	$73	$73	$73
EC2 Nodes (3×t3.medium)	$100	$180	$250
RDS PostgreSQL	$280	$420	$600
S3 Storage (100GB)	$2.30	$2.30	$2.30
Data Transfer	$10	$30	$100
TOTAL	**$466**	**$705**	**$1,025**

VPC Network Design

```
AWS Region: ap-south-1 (Mumbai)
├── VPC CIDR: 10.0.0.0/16
├── Public Subnets (ALB)
│   ├── 10.0.1.0/24 (ap-south-1a)
│   └── 10.0.2.0/24 (ap-south-1b)
├── Private Subnets (EKS Nodes)
│   ├── 10.0.11.0/24 (ap-south-1a)
│   └── 10.0.12.0/24 (ap-south-1b)
├── Database Subnets (RDS)
│   ├── 10.0.21.0/24 (ap-south-1a)
```

```
│   └── 10.0.22.0/24 (ap-south-1b)
└── NAT Gateway for private subnets
```

3. Docker Strategy & Image Optimization

3.1 NestJS API Dockerfile (Multi-Stage Build)

```
# Stage 1: Builder
FROM node:18-alpine AS builder

WORKDIR /app

# Copy package files
COPY package*.json ./
RUN npm ci --only=production && \\
    npm cache clean --force

# Copy source code
COPY . .

# Build NestJS
RUN npm run build

# Stage 2: Runtime
FROM node:18-alpine

WORKDIR /app

# Install dumb-init for proper signal handling
RUN apk add --no-cache dumb-init

# Create non-root user
RUN addgroup -g 1001 -S nodejs && \\
    adduser -S nestjs -u 1001

# Copy only necessary files from builder
COPY --from=builder --chown=nestjs:nodejs /app/node_modules ./node_modul
```

```
COPY --from=builder --chown=nestjs:nodejs /app/dist ./dist
COPY --from=builder --chown=nestjs:nodejs /app/package*.json ./

# Switch to non-root user
USER nestjs

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD node -e "require('http').get('http://localhost:3000/health', (r)

# Expose port
EXPOSE 3000

# Use dumb-init to handle signals properly
ENTRYPOINT ["dumb-init", "--"]

# Start application
CMD ["node", "dist/main.js"]
```

3.2 Angular Frontend Dockerfile

```
# Stage 1: Build
FROM node:18-alpine AS builder

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .
RUN npm run build -- --configuration production

# Stage 2: Serve with nginx
FROM nginx:alpine

# Remove default config
RUN rm -rf /etc/nginx/conf.d/*

# Copy nginx config
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

```
# Copy built Angular app from builder
COPY --from=builder /app/dist/angular-app /usr/share/nginx/html

# Create non-root user for nginx
RUN addgroup -g 1001 -S www && \\
    adduser -S www -u 1001 && \\
    chown -R www:www /usr/share/nginx/html

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD wget --no-verbose --tries=1 --spider http://localhost/health ||

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

3.3 nginx.conf for Angular

```
server {
    listen 80;
    server_name _;

    root /usr/share/nginx/html;
    index index.html index.htm;

    # Gzip compression
    gzip on;
    gzip_types text/plain text/css text/javascript application/javascrip
    gzip_min_length 1000;

    # Cache control
    location ~* \.(?:bundle\.js|bundle\.css)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    location ~* \.(js|css|fonts)$ {
        expires 30d;
        add_header Cache-Control "public";
    }

    # API proxy
```

```
    location /api/ {
        proxy_pass http://nestjs-api.default.svc.cluster.local:3000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # SPA routing
    location / {
        try_files $uri $uri/ /index.html;
        expires -1;
        add_header Cache-Control "public, max-age=0";
    }

    # Health check
    location /health {
        access_log off;
        return 200 "healthy\n";
        add_header Content-Type text/plain;
    }
}
```

3.4 Docker Image Optimization Tips

- Use Alpine Linux for smaller images (5-10x smaller)
- Multi-stage builds to exclude build dependencies
- Run as non-root user for security
- Include health checks for container orchestration
- Use .dockerignore to exclude unnecessary files
- Layer caching: place stable commands first
- Compress images: ~80-150MB for NestJS, ~50-80MB for Angular

4. Kubernetes Architecture & EKS Design

4.1 EKS Cluster Architecture

```
AWS EKS Cluster (ap-south-1)
├── Control Plane (Managed by AWS)
│   ├── API Server
│   ├── etcd
│   ├── Scheduler
│   └── Controller Manager
├── Worker Nodes (EC2)
│   ├── Node 1 (ap-south-1a, t3.large)
│   ├── Node 2 (ap-south-1b, t3.large)
│   └── Node 3 (ap-south-1b, t3.medium)
├── Add-ons
│   ├── VPC CNI for networking
│   ├── CoreDNS for service discovery
│   ├── kube-proxy for load balancing
│   └── AWS Load Balancer Controller
└── Third-party
    ├── Prometheus (monitoring)
    ├── Loki (logging)
    ├── ArgoCD (GitOps)
    └── Cert-Manager (TLS)
```

4.2 Kubernetes Namespace Structure

```
# Create namespaces
kubectl create namespace production
kubectl create namespace staging
kubectl create namespace monitoring
kubectl create namespace ingress-nginx
```

```
# Apply namespace labels for pod security
kubectl label namespace production pod-security.kubernetes.io/enforce=ba
```

4.3 NestJS API Deployment (Kubernetes)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nestjs-api
  namespace: production
  labels:
    app: nestjs-api
    tier: backend
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: nestjs-api
  template:
    metadata:
      labels:
        app: nestjs-api
        tier: backend
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "3000"
        prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: nestjs-api
      securityContext:
        runAsNonRoot: true
        runAsUser: 1001
        fsGroup: 1001
      containers:
      - name: nestjs-api
        image: 123456789.dkr.ecr.ap-south-1.amazonaws.com/nestjs-api:1.0
        imagePullPolicy: IfNotPresent
```

```
        ports:
        - name: http
          containerPort: 3000
          protocol: TCP
        env:
        - name: NODE_ENV
          value: "production"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: connection-string
        - name: AWS_S3_BUCKET
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: s3-bucket
        - name: AWS_REGION
          value: "ap-south-1"

        # Health checks
        livenessProbe:
          httpGet:
            path: /health
            port: http
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3

        readinessProbe:
          httpGet:
            path: /ready
            port: http
          initialDelaySeconds: 10
          periodSeconds: 5
          timeoutSeconds: 3
          failureThreshold: 3

        # Resource limits
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
```

```
            memory: "512Mi"
            cpu: "500m"

        # Volume mounts
        volumeMounts:
        - name: config
          mountPath: /app/config
          readOnly: true

        # Security context
        securityContext:
          allowPrivilegeEscalation: false
          readOnlyRootFilesystem: true
          capabilities:
            drop:
            - ALL

      # Pod anti-affinity for spreading
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - nestjs-api
              topologyKey: kubernetes.io/hostname

      volumes:
      - name: config
        configMap:
          name: nestjs-config

---
apiVersion: v1
kind: Service
metadata:
  name: nestjs-api
  namespace: production
spec:
  type: ClusterIP
  selector:
```

```
    app: nestjs-api
  ports:
  - name: http
    port: 3000
    targetPort: http
    protocol: TCP

---
apiVersion: autoscaling.k8s.io/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nestjs-api-hpa
  namespace: production
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nestjs-api
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 50
        periodSeconds: 15
    scaleUp:
      stabilizationWindowSeconds: 0
      policies:
      - type: Percent
        value: 100
```

```
        periodSeconds: 15
      - type: Pods
        value: 2
        periodSeconds: 15
      selectPolicy: Max
```

4.4 Angular Frontend Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: angular-frontend
  namespace: production
spec:
  replicas: 2
  selector:
    matchLabels:
      app: angular-frontend
  template:
    metadata:
      labels:
        app: angular-frontend
    spec:
      containers:
      - name: angular
        image: 123456789.dkr.ecr.ap-south-1.amazonaws.com/angular-fronte
        ports:
        - containerPort: 80
        livenessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 20
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
        resources:
          requests:
```

```
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "200m"

---
apiVersion: v1
kind: Service
metadata:
  name: angular-frontend
  namespace: production
spec:
  type: LoadBalancer
  selector:
    app: angular-frontend
  ports:
  - port: 80
    targetPort: 80

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: production
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - app.example.com
    secretName: app-tls
  rules:
  - host: app.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: angular-frontend
            port:
```

```
              number: 80
      - path: /api
        pathType: Prefix
        backend:
          service:
            name: nestjs-api
            port:
              number: 3000
```

5. Database Setup (RDS PostgreSQL)

5.1 RDS PostgreSQL Creation (Terraform)

```
resource "aws_db_instance" "main" {
  identifier     = "app-production-db"
  engine         = "postgres"
  engine_version = "15.4"
  instance_class = "db.t3.large"

  allocated_storage     = 100
  max_allocated_storage = 500
  storage_type          = "gp3"
  storage_encrypted     = true

  db_name  = "production"
  username = "postgres"
  password = random_password.db_password.result

  # Availability
  multi_az               = true
  publicly_accessible    = false
  db_subnet_group_name   = aws_db_subnet_group.main.name
  vpc_security_group_ids = [aws_security_group.rds.id]

  # Backups
  backup_retention_period = 30
  backup_window           = "03:00-04:00"
  copy_tags_to_snapshot   = true
  skip_final_snapshot     = false
```

```
  # Performance
  performance_insights_enabled = true
  enabled_cloudwatch_logs_exports = ["postgresql"]
  deletion_protection = true

  tags = {
    Name = "app-production-db"
    Environment = "production"
  }
}

resource "aws_secretsmanager_secret" "db_password" {
  name = "app/db/credentials"
}

resource "aws_secretsmanager_secret_version" "db_password" {
  secret_id = aws_secretsmanager_secret.db_password.id
  secret_string = jsonencode({
    username = aws_db_instance.main.username
    password = aws_db_instance.main.password
    engine   = "postgres"
    host     = aws_db_instance.main.endpoint
    port     = 5432
    dbname   = aws_db_instance.main.db_name
  })
}
```

5.2 NestJS Database Configuration

```
// src/database/database.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { ConfigService } from '@nestjs/config';

@Module({
  imports: [
    TypeOrmModule.forRootAsync({
      inject: [ConfigService],
      useFactory: (config: ConfigService) => ({
        type: 'postgres',
        host: config.get('DB_HOST'),
```

```
        port: config.get('DB_PORT', 5432),
        username: config.get('DB_USER'),
        password: config.get('DB_PASSWORD'),
        database: config.get('DB_NAME'),
        entities: [__dirname + '/../**/*.entity.{js,ts}'],
        synchronize: false,
        migrations: [__dirname + '/../migrations/*.{js,ts}'],
        migrationsRun: true,
        logging: config.get('NODE_ENV') === 'development',
        ssl: {
          rejectUnauthorized: false,
        },
        pool: {
          min: 5,
          max: 20,
        },
      }),
    }),
  ],
})
export class DatabaseModule {}
```

5.3 Database Migrations (TypeORM)

```
// src/migrations/1704067200000-CreateUsersTable.ts
import { MigrationInterface, QueryRunner, Table } from 'typeorm';

export class CreateUsersTable1704067200000 implements MigrationInterface
  public async up(queryRunner: QueryRunner): Promise {
    await queryRunner.createTable(
      new Table({
        name: 'users',
        columns: [
          {
            name: 'id',
            type: 'uuid',
            isPrimary: true,
            default: 'gen_random_uuid()',
          },
          {
            name: 'email',
            type: 'varchar',
```

```
            isUnique: true,
            isNullable: false,
          },
          {
            name: 'password_hash',
            type: 'varchar',
            isNullable: false,
          },
          {
            name: 'first_name',
            type: 'varchar',
            isNullable: true,
          },
          {
            name: 'last_name',
            type: 'varchar',
            isNullable: true,
          },
          {
            name: 'is_active',
            type: 'boolean',
            default: true,
          },
          {
            name: 'created_at',
            type: 'timestamp',
            default: 'CURRENT_TIMESTAMP',
          },
          {
            name: 'updated_at',
            type: 'timestamp',
            default: 'CURRENT_TIMESTAMP',
            onUpdate: 'CURRENT_TIMESTAMP',
          },
        ],
        indices: [
          { columnNames: ['email'] },
          { columnNames: ['created_at'] },
        ],
      }),
    );
  }

  public async down(queryRunner: QueryRunner): Promise {
    await queryRunner.dropTable('users');
```

```
  }
}
```

5.4 PostgreSQL Performance Tuning

Parameter	Value	Purpose
shared_buffers	4GB (25% of RAM)	Cached data in memory
effective_cache_size	12GB (75% of RAM)	Query planner estimates
work_mem	64MB	Sort/hash memory per operation
maintenance_work_mem	1GB	VACUUM/INDEX memory
max_connections	200	Maximum concurrent connections
random_page_cost	1.1	SSD optimization

6. S3 Storage Configuration

6.1 S3 Bucket Setup (Terraform)

```
resource "aws_s3_bucket" "app_storage" {
  bucket = "app-storage-prod-${random_id.bucket.hex}"

  tags = {
    Name = "app-storage"
    Environment = "production"
  }
```

```
}

# Enable versioning
resource "aws_s3_bucket_versioning" "app_storage" {
  bucket = aws_s3_bucket.app_storage.id
  versioning_configuration {
    status = "Enabled"
  }
}

# Block public access
resource "aws_s3_bucket_public_access_block" "app_storage" {
  bucket = aws_s3_bucket.app_storage.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

# Encryption
resource "aws_s3_bucket_server_side_encryption_configuration" "app_stora
  bucket = aws_s3_bucket.app_storage.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

# Lifecycle policy
resource "aws_s3_bucket_lifecycle_configuration" "app_storage" {
  bucket = aws_s3_bucket.app_storage.id

  rule {
    id     = "delete-old-versions"
    status = "Enabled"
    noncurrent_version_expiration {
      noncurrent_days = 90
    }
  }

  rule {
    id     = "intelligent-tiering"
```

```
    status = "Enabled"
    transition {
      days          = 30
      storage_class = "INTELLIGENT_TIERING"
    }
  }
}

# CORS policy
resource "aws_s3_bucket_cors_configuration" "app_storage" {
  bucket = aws_s3_bucket.app_storage.id

  cors_rule {
    allowed_headers = ["*"]
    allowed_methods = ["GET", "PUT", "POST", "DELETE"]
    allowed_origins = ["https://app.example.com"]
    expose_headers  = ["ETag"]
    max_age_seconds = 3000
  }
}

# IAM policy for pod access
resource "aws_iam_role" "app_pod_role" {
  name = "app-pod-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Federated = "arn:aws:iam::ACCOUNT_ID:oidc-provider/oidc.eks.ap-s
      }
      Condition = {
        StringEquals = {
          "oidc.eks.ap-south-1.amazonaws.com/id/EXAMPLEID:sub" = "system
        }
      }
    }]
  })
}

resource "aws_iam_role_policy" "s3_access" {
  name = "s3-access"
  role = aws_iam_role.app_pod_role.id
```

```
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Action = [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListBucket"
      ]
      Resource = [
        aws_s3_bucket.app_storage.arn,
        "${aws_s3_bucket.app_storage.arn}/*"
      ]
    }]
  })
}
```

6.2 NestJS S3 File Upload Service

```
// src/storage/s3.service.ts
import { Injectable } from '@nestjs/common';
import { S3Client, PutObjectCommand, GetObjectCommand, DeleteObjectComma
import { getSignedUrl } from '@aws-sdk/s3-request-presigner';
import { ConfigService } from '@nestjs/config';

@Injectable()
export class S3Service {
  private s3Client: S3Client;

  constructor(private config: ConfigService) {
    this.s3Client = new S3Client({
      region: this.config.get('AWS_REGION'),
      credentials: {
        accessKeyId: this.config.get('AWS_ACCESS_KEY_ID'),
        secretAccessKey: this.config.get('AWS_SECRET_ACCESS_KEY'),
      },
    });
  }

  async uploadFile(
```

```
    bucket: string,
    key: string,
    body: Buffer,
    contentType: string,
  ): Promise {
    const command = new PutObjectCommand({
      Bucket: bucket,
      Key: key,
      Body: body,
      ContentType: contentType,
    });

    await this.s3Client.send(command);
    return `s3://${bucket}/${key}`;
  }

  async getPresignedUrl(bucket: string, key: string): Promise {
    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    return getSignedUrl(this.s3Client, command, { expiresIn: 3600 });
  }

  async deleteFile(bucket: string, key: string): Promise {
    const command = new DeleteObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    await this.s3Client.send(command);
  }
}
```

6.3 Angular File Upload Component

```
// src/app/components/file-upload/file-upload.component.ts
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Subject } from 'rxjs';
```

```
@Component({
  selector: 'app-file-upload',
  template: `

    <div class="upload-area">
      <input type="file" #fileinput="" (change)="onFileSelected($event)"
      <button (click)="upload()" [disabled]="!selectedFile || uploading"
        {{ '{{' }} uploading ? 'Uploading...' : 'Upload' {{ '}}' }}
      </button>
      <div class="progress" *ngif="uploadProgress$ | async as progress">
        <div class="bar" [style.width.%]="progress"></div>
      </div>
    </div>

  `,
})
export class FileUploadComponent {
  selectedFile: File | null = null;
  uploadProgress$ = new Subject();
  uploading = false;

  constructor(private http: HttpClient) {}

  onFileSelected(event: Event): void {
    const target = event.target as HTMLInputElement;
    this.selectedFile = target.files?.[0] || null;
  }

  upload(): void {
    if (!this.selectedFile) return;

    const formData = new FormData();
    formData.append('file', this.selectedFile);

    this.uploading = true;

    this.http.post('/api/upload', formData, {
      reportProgress: true,
      observe: 'events',
    }).subscribe(
      (event: any) => {
        if (event.type === 4) {
          this.uploading = false;
          console.log('Upload complete:', event.body);
        } else if (event.type === 1) {
```

```
            const progress = Math.round((event.loaded / event.total) * 100
            this.uploadProgress$.next(progress);
          }
        },
        (error) => {
          this.uploading = false;
          console.error('Upload failed:', error);
        }
      );
    }
  }
```

7. NestJS API Development Setup

7.1 Project Initialization

```
# Initialize NestJS project
nest new nestjs-api
cd nestjs-api

# Install essential dependencies
npm install @nestjs/common @nestjs/core @nestjs/platform-express
npm install @nestjs/typeorm typeorm pg
npm install @nestjs/config
npm install @nestjs/jwt @nestjs/passport passport passport-jwt
npm install aws-sdk @aws-sdk/client-s3
npm install class-validator class-transformer
npm install @nestjs/swagger swagger-ui-express
npm install --save-dev @types/node typescript

# Create directory structure
mkdir src/{modules,common,config,decorators,guards,interceptors,pipes}
```

7.2 NestJS App Module (Main Entry)

```
// src/app.module.ts
import { Module, MiddlewareConsumer, NestModule } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { TypeOrmModule } from '@nestjs/typeorm';
import { APP_FILTER, APP_INTERCEPTOR } from '@nestjs/core';
import { LoggerMiddleware } from './common/middleware/logger.middleware'
import { LoggingInterceptor } from './common/interceptors/logging.interc
import { AllExceptionsFilter } from './common/filters/all-exceptions.fil
import { DatabaseModule } from './modules/database/database.module';
import { AuthModule } from './modules/auth/auth.module';
import { UsersModule } from './modules/users/users.module';
import { FilesModule } from './modules/files/files.module';
import { HealthModule } from './modules/health/health.module';

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      envFilePath: '.env',
    }),
    DatabaseModule,
    AuthModule,
    UsersModule,
    FilesModule,
    HealthModule,
  ],
  providers: [
    {
      provide: APP_FILTER,
      useClass: AllExceptionsFilter,
    },
    {
      provide: APP_INTERCEPTOR,
      useClass: LoggingInterceptor,
    },
  ],
})
export class AppModule implements NestModule {
  configure(consumer: MiddlewareConsumer) {
    consumer.apply(LoggerMiddleware).forRoutes('*');
  }
}
```

7.3 Authentication Module

```
// src/modules/auth/auth.service.ts
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';
import { UsersService } from '../users/users.service';

@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService,
  ) {}

  async validateUser(email: string, password: string) {
    const user = await this.usersService.findByEmail(email);
    if (user && await bcrypt.compare(password, user.passwordHash)) {
      const { passwordHash, ...result } = user;
      return result;
    }
    return null;
  }

  async login(user: any) {
    const payload = { sub: user.id, email: user.email };
    return {
      access_token: this.jwtService.sign(payload, {
        expiresIn: '1h',
      }),
      refresh_token: this.jwtService.sign(payload, {
        expiresIn: '7d',
      }),
    };
  }

  async register(email: string, password: string) {
    const hashedPassword = await bcrypt.hash(password, 10);
    return this.usersService.create({
      email,
      passwordHash: hashedPassword,
    });
  }
```

```
}

// src/modules/auth/auth.controller.ts
import { Controller, Post, Body, UseGuards, Get, Req } from '@nestjs/com
import { AuthService } from './auth.service';
import { JwtAuthGuard } from '../../common/guards/jwt-auth.guard';
import { LocalAuthGuard } from '../../common/guards/local-auth.guard';
import { RegisterDto } from './dto/register.dto';

@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @Post('register')
  async register(@Body() registerDto: RegisterDto) {
    return this.authService.register(registerDto.email, registerDto.pass
  }

  @UseGuards(LocalAuthGuard)
  @Post('login')
  async login(@Req() req: any) {
    return this.authService.login(req.user);
  }

  @UseGuards(JwtAuthGuard)
  @Get('profile')
  getProfile(@Req() req: any) {
    return req.user;
  }
}
```

7.4 Health Check Endpoint

```
// src/modules/health/health.controller.ts
import { Controller, Get } from '@nestjs/common';
import { HealthCheckService, HttpHealthIndicator, TypeOrmHealthIndicator

@Controller()
export class HealthController {
  constructor(
    private health: HealthCheckService,
    private db: TypeOrmHealthIndicator,
```

```
    private http: HttpHealthIndicator,
  ) {}

  @Get('health')
  check() {
    return this.health.check([
      () => this.db.pingCheck('database', { timeout: 1500 }),
    ]);
  }

  @Get('ready')
  ready() {
    return { status: 'ready' };
  }
}
```

7.5 .env Configuration

```
NODE_ENV=production
PORT=3000

# Database
DATABASE_URL=postgresql://user:password@host:5432/dbname
DB_HOST=postgres.example.com
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=secure_password
DB_NAME=production

# JWT
JWT_SECRET=your-secret-key-min-32-chars-long
JWT_EXPIRATION=3600

# AWS
AWS_REGION=ap-south-1
AWS_ACCESS_KEY_ID=AKIA...
AWS_SECRET_ACCESS_KEY=...
AWS_S3_BUCKET=app-storage-bucket

# CORS
CORS_ORIGIN=https://app.example.com
```

```
# Logging
LOG_LEVEL=info
```

8. Angular Frontend Development

8.1 Angular Project Setup

```
# Create Angular project
ng new angular-frontend
cd angular-frontend

# Install dependencies
ng add @angular/material
npm install @angular/common @angular/platform-browser-dynamic
npm install rxjs tslib zone.js
npm install ngrx ngrx-store-localstorage
npm install ngx-translate @ngx-translate/core
npm install http-interceptor

# Generate modules/components
ng generate module modules/auth
ng generate component modules/auth/components/login
ng generate component modules/auth/components/register
ng generate service services/auth

ng generate module modules/dashboard
ng generate component modules/dashboard/pages/dashboard

ng generate module modules/files
ng generate component modules/files/components/upload
```

8.2 Angular Module Structure

```
// src/app/app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from '@angular/platform-browser/anima
```

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/htt
import { StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { JwtInterceptor } from './interceptors/jwt.interceptor';
import { ErrorInterceptor } from './interceptors/error.interceptor';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule,
    AppRoutingModule,
    StoreModule.forRoot({}),
    EffectsModule.forRoot([]),
  ],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: JwtInterceptor,
      multi: true,
    },
    {
      provide: HTTP_INTERCEPTORS,
      useClass: ErrorInterceptor,
      multi: true,
    },
  ],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

8.3 Authentication Service (RxJS)

```
// src/app/services/auth.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable, of } from 'rxjs';
import { map, catchError, tap, switchMap } from 'rxjs/operators';
```

```
interface User {
  id: string;
  email: string;
  firstName: string;
  lastName: string;
}

interface AuthResponse {
  access_token: string;
  refresh_token: string;
}

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private currentUserSubject: BehaviorSubject;
  public currentUser$: Observable;

  constructor(private http: HttpClient) {
    this.currentUserSubject = new BehaviorSubject(
      this.getUserFromStorage(),
    );
    this.currentUser$ = this.currentUserSubject.asObservable();
  }

  public get currentUserValue(): User | null {
    return this.currentUserSubject.value;
  }

  login(email: string, password: string): Observable {
    return this.http.post('/api/auth/login', {
      email,
      password,
    }).pipe(
      tap((response) => {
        this.setTokens(response);
        this.fetchCurrentUser();
      }),
    );
  }

  register(email: string, password: string): Observable {
    return this.http.post('/api/auth/register', {
```

```
      email,
      password,
    });
  }

  logout(): void {
    localStorage.removeItem('access_token');
    localStorage.removeItem('refresh_token');
    this.currentUserSubject.next(null);
  }

  private fetchCurrentUser(): void {
    this.http.get('/api/auth/profile').subscribe({
      next: (user) => this.currentUserSubject.next(user),
      error: () => this.logout(),
    });
  }

  private setTokens(response: AuthResponse): void {
    localStorage.setItem('access_token', response.access_token);
    localStorage.setItem('refresh_token', response.refresh_token);
  }

  private getUserFromStorage(): User | null {
    const user = localStorage.getItem('user');
    return user ? JSON.parse(user) : null;
  }
}

// src/app/interceptors/jwt.interceptor.ts
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@a
import { Observable } from 'rxjs';
import { AuthService } from '../services/auth.service';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}

  intercept(
    request: HttpRequest,
    next: HttpHandler,
  ): Observable> {
    const token = localStorage.getItem('access_token');
```

```
    if (token) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${token}`,
        },
      });
    }

    return next.handle(request);
  }
}
```

8.4 Dashboard Component with State Management

```
// src/app/modules/dashboard/components/dashboard.component.ts
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { AuthService } from '../../../services/auth.service';

@Component({
  selector: 'app-dashboard',
  template: `
    <div class="dashboard">
      <header>
        <h1>Dashboard</h1>
        <div *ngif="currentUser$ | async as user">
          Welcome, {{ user.firstName }}!
        </div>
      </header>

      <main>
        <div class="widgets">
          <div class="card">
            <h3>Stats</h3>
            <p>Your statistics here</p>
          </div>

          <app-file-upload></app-file-upload>
        </div>
      </main>
    </div>
  `,
```

```
  styles: [`
    .dashboard {
      padding: 20px;
    }

    header {
      display: flex;
      justify-content: space-between;
      align-items: center;
      margin-bottom: 30px;
      border-bottom: 1px solid #ccc;
      padding-bottom: 15px;
    }

    .widgets {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
      gap: 20px;
    }

    .card {
      background: white;
      border-radius: 8px;
      padding: 20px;
      box-shadow: 0 2px 8px rgba(0,0,0,0.1);
    }
  `],
})
export class DashboardComponent implements OnInit {
  currentUser$: Observable;

  constructor(private authService: AuthService) {
    this.currentUser$ = this.authService.currentUser$;
  }

  ngOnInit(): void {}
}
```

9. CI/CD Pipeline & GitOps

9.1 GitHub Actions Workflow

```
# .github/workflows/build-deploy.yml
name: Build and Deploy to EKS

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v2
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ap-south-1

    - name: Login to ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v1

    - name: Build NestJS API image
      working-directory: ./api
      run: |
        docker build -t nestjs-api:latest .
        docker tag nestjs-api:latest ${{ steps.login-ecr.outputs.registr
        docker push ${{ steps.login-ecr.outputs.registry }}/nestjs-api:$

    - name: Build Angular Frontend image
      working-directory: ./frontend
      run: |
        docker build -t angular-frontend:latest .
        docker tag angular-frontend:latest ${{ steps.login-ecr.outputs.r
        docker push ${{ steps.login-ecr.outputs.registry }}/angular-fron
```

```
    - name: Update kube manifests
      run: |
        sed -i "s|IMAGE_TAG|${{ github.sha }}|g" k8s/overlays/productior

    - name: Commit and push
      run: |
        git config user.name "GitHub Actions"
        git config user.email "actions@github.com"
        git add k8s/
        git commit -m "Update image tags to ${{ github.sha }}"
        git push

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v3

    - name: Deploy with ArgoCD
      run: |
        curl -X POST ${{ secrets.ARGOCD_SERVER }}/api/v1/applications/ap
          -H "Authorization: Bearer ${{ secrets.ARGOCD_TOKEN }}"
```

9.2 ArgoCD GitOps Setup

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/yourorg/app
    targetRevision: main
    path: k8s/overlays/production
  destination:
    server: https://kubernetes.default.svc
    namespace: production
  syncPolicy:
```

```
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

9.3 Kustomize Overlay Structure

```
# k8s/overlays/production/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

namespace: production

bases:
- ../../base

commonLabels:
  app.kubernetes.io/part-of: full-stack-app
  app.kubernetes.io/environment: production

commonAnnotations:
  deployment.kubernetes.io/revision: "1"

replicas:
- name: nestjs-api
  count: 3
- name: angular-frontend
  count: 2

images:
- name: nestjs-api
  newTag: IMAGE_TAG
- name: angular-frontend
  newTag: IMAGE_TAG

patchesStrategicMerge:
- deployment-patch.yaml

resources:
- namespace.yaml
- configmap.yaml
```

```
- secrets.yaml
- ingress.yaml
```

10. Deployment to EKS

10.1 EKS Cluster Creation

```
# Create EKS cluster with eksctl
eksctl create cluster \\
  --name app-production \\
  --region ap-south-1 \\
  --nodegroup-name standard-nodes \\
  --node-type t3.large \\
  --nodes 3 \\
  --nodes-min 2 \\
  --nodes-max 10 \\
  --managed \\
  --enable-ssm

# Get kubeconfig
aws eks update-kubeconfig \\
  --region ap-south-1 \\
  --name app-production

# Verify cluster
kubectl get nodes
kubectl get pods --all-namespaces
```

10.2 Install Required Add-ons

```
# Install NGINX Ingress Controller
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm install nginx-ingress ingress-nginx/ingress-nginx \\
  --namespace ingress-nginx \\
  --create-namespace
```

```
# Install Cert-Manager
helm repo add jetstack https://charts.jetstack.io
helm repo update
helm install cert-manager jetstack/cert-manager \\
  --namespace cert-manager \\
  --create-namespace \\
  --set installCRDs=true

# Install Metrics Server (for HPA)
kubectl apply -f https://github.com/kubernetes-sigs/metrics-
server/releases/latest/download/components.yaml

# Install Prometheus for monitoring
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install prometheus prometheus-community/kube-prometheus-stack \\
  --namespace monitoring \\
  --create-namespace

# Install Loki for logging
helm repo add grafana https://grafana.github.io/helm-charts
helm install loki grafana/loki-stack \\
  --namespace monitoring \\
  --set loki.persistence.enabled=true
```

10.3 Deploy Application

```
# Create namespace
kubectl create namespace production

# Create secrets
kubectl create secret generic db-credentials \\
  --from-literal=connection-string="postgresql://user:pass@host/db" \\
  -n production

# Create ConfigMap
kubectl create configmap app-config \\
  --from-literal=s3-bucket="app-storage-bucket" \\
  -n production

# Apply manifests
kubectl apply -f k8s/overlays/production/

# Verify deployment
```

```
kubectl get deployments -n production
kubectl get pods -n production
kubectl get services -n production

# Check logs
kubectl logs -f deployment/nestjs-api -n production
```

10.4 SSL Certificate with Cert-Manager

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: admin@example.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: production
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - app.example.com
    - api.example.com
    secretName: app-tls
  rules:
  - host: app.example.com
```

```
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: angular-frontend
            port:
              number: 80
  - host: api.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nestjs-api
            port:
              number: 3000
```

11. Monitoring, Logging & Observability

11.1 Prometheus Metrics

```
// src/common/interceptors/metrics.interceptor.ts
import {
  Injectable,
  NestInterceptor,
  ExecutionContext,
} from '@nestjs/common';
import { Observable } from 'rxjs';
import { tap } from 'rxjs/operators';
import { register, Counter, Histogram } from 'prom-client';

@Injectable()
export class MetricsInterceptor implements NestInterceptor {
  private requestCounter = new Counter({
    name: 'http_requests_total',
    help: 'Total HTTP requests',
```

```
    labelNames: ['method', 'route', 'status'],
  });

  private requestDuration = new Histogram({
    name: 'http_request_duration_seconds',
    help: 'HTTP request duration',
    labelNames: ['method', 'route'],
    buckets: [0.1, 0.5, 1, 2, 5],
  });

  intercept(context: ExecutionContext, next: any): Observable {
    const request = context.switchToHttp().getRequest();
    const { method, url } = request;
    const start = Date.now();

    return next.handle().pipe(
      tap(
        () => {
          const duration = (Date.now() - start) / 1000;
          const status = context.switchToHttp().getResponse().statusCode

          this.requestCounter.inc({
            method,
            route: url,
            status,
          });

          this.requestDuration.observe({ method, route: url }, duration)
        },
        (error) => {
          const duration = (Date.now() - start) / 1000;
          this.requestCounter.inc({
            method,
            route: url,
            status: 500,
          });

          this.requestDuration.observe({ method, route: url }, duration)
        },
      ),
    );
  }
}

// Expose metrics endpoint
```

```
@Controller()
export class MetricsController {
  @Get('/metrics')
  metrics() {
    return register.metrics();
  }
}
```

11.2 CloudWatch Logs Integration

```
// src/common/logger/logger.service.ts
import { Injectable, Logger as NestLogger } from '@nestjs/common';
import { CloudWatchClient, PutLogEventsCommand } from '@aws-sdk/client-c

@Injectable()
export class LoggerService extends NestLogger {
  private cloudwatch: CloudWatchClient;
  private logGroupName = '/ecs/app';
  private logStreamName = `${new Date().toISOString().split('T')[0]}-str

  constructor() {
    super();
    this.cloudwatch = new CloudWatchClient({ region: 'ap-south-1' });
  }

  log(message: string, context?: string): void {
    super.log(message, context);
    this.sendToCloudWatch('INFO', message, context);
  }

  error(message: string, trace?: string, context?: string): void {
    super.error(message, trace, context);
    this.sendToCloudWatch('ERROR', message, context, trace);
  }

  warn(message: string, context?: string): void {
    super.warn(message, context);
    this.sendToCloudWatch('WARN', message, context);
  }

  private async sendToCloudWatch(
    level: string,
```

```
    message: string,
    context?: string,
    trace?: string,
  ): Promise {
    try {
      const logMessage = {
        timestamp: new Date().getTime(),
        level,
        context,
        message,
        trace,
      };

      const command = new PutLogEventsCommand({
        logGroupName: this.logGroupName,
        logStreamName: this.logStreamName,
        logEvents: [
          {
            message: JSON.stringify(logMessage),
            timestamp: new Date().getTime(),
          },
        ],
      });

      await this.cloudwatch.send(command);
    } catch (error) {
      console.error('Failed to send logs to CloudWatch:', error);
    }
  }
}
```

11.3 Grafana Dashboard Setup

Key Metrics to Monitor:

- Pod CPU & Memory usage
- HTTP request rate and latency (P50, P95, P99)
- Database connection pool usage
- Error rate by endpoint

- Pod restart count
- Network I/O

12. Security & Network Policies

12.1 Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: nestjs-api-netpol
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: nestjs-api
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: angular-frontend
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 3000
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: postgres
    ports:
    - protocol: TCP
```

```
      port: 5432
  - to:
    - namespaceSelector: {}
    ports:
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53
```

12.2 Pod Security Policy

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted-psp
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities:
  - ALL
  volumes:
  - 'configMap'
  - 'emptyDir'
  - 'projected'
  - 'secret'
  - 'downwardAPI'
  - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: 'MustRunAsNonRoot'
  seLinux:
    rule: 'MustRunAs'
    seLinuxOptions:
      level: "s0:c123,c456"
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
```

12.3 RBAC Configuration

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nestjs-api
  namespace: production

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: nestjs-api-role
  namespace: production
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: nestjs-api-rolebinding
  namespace: production
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: nestjs-api-role
subjects:
- kind: ServiceAccount
  name: nestjs-api
  namespace: production
```

13. Scaling & Performance Optimization

13.1 Horizontal Pod Autoscaling (HPA)

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nestjs-api-hpa
  namespace: production
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nestjs-api
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  - type: Pods
    pods:
      metric:
        name: http_requests_per_second
      target:
        type: AverageValue
        averageValue: "1000"
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 50
        periodSeconds: 15
    scaleUp:
      stabilizationWindowSeconds: 0
```

```
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15
      - type: Pods
        value: 4
        periodSeconds: 15
      selectPolicy: Max
```

13.2 Load Testing with k6

```
// load-test.js
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '30s', target: 20 },
    { duration: '1m', target: 50 },
    { duration: '30s', target: 0 },
  ],
};

export default function () {
  const res = http.get('http://api.example.com/api/users');

  check(res, {
    'status is 200': (r) => r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });

  sleep(1);
}

// Run with:
// k6 run load-test.js
```

13.3 Vertical Pod Autoscaling (VPA)

```
# Install VPA
git clone https://github.com/kubernetes/autoscaler.git
cd autoscaler/vertical-pod-autoscaler
./hack/vpa-up.sh

# Apply VPA resource
kubectl apply -f - <
```

13.4 Caching Strategy

```
// src/modules/users/users.service.ts
import { Injectable, Inject } from '@nestjs/common';
import { CACHE_MANAGER } from '@nestjs/cache-manager';
import { Cache } from 'cache-manager';
import { Repository } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from './user.entity';

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User)
    private usersRepository: Repository,
    @Inject(CACHE_MANAGER) private cacheManager: Cache,
  ) {}

  async findById(id: string): Promise {
    const cacheKey = `user:${id}`;

    // Try cache first
    let user = await this.cacheManager.get(cacheKey);

    if (!user) {
      // Cache miss - query database
      user = await this.usersRepository.findOneBy({ id });

      if (user) {
        // Cache for 5 minutes
        await this.cacheManager.set(cacheKey, user, 300000);
      }
    }
```

```
    return user || null;
  }

  async update(id: string, data: Partial): Promise {
    const user = await this.usersRepository.findOneBy({ id });

    if (user) {
      Object.assign(user, data);
      await this.usersRepository.save(user);

      // Invalidate cache
      await this.cacheManager.del(`user:${id}`);
    }

    return user;
  }
}
```

14. Troubleshooting & Best Practices

14.1 Common Issues & Solutions

Issue	Symptoms	Solution
Pods pending	kubectl describe pod shows pending	Check node capacity: `kubectl top nodes`
DB connection timeout	Connection pool exhausted	Increase max_connections or pool size
Pod OOMKilled	Memory limit exceeded	Increase memory limits in deployment

Issue	Symptoms	Solution
Image pull errors	ImagePullBackOff status	Verify ECR credentials and image path
Service unreachable	Timeout connecting to service	Check NetworkPolicy and security groups
High latency	Requests taking >2 seconds	Check CPU usage, database queries, add more replicas

14.2 Debugging Commands

```
# Check pod status
kubectl describe pod POD_NAME -n production

# View logs
kubectl logs -f deployment/nestjs-api -n production

# Execute command in pod
kubectl exec -it POD_NAME -n production -- /bin/sh

# Port forward
kubectl port-forward service/nestjs-api 3000:3000 -n production

# Check resource usage
kubectl top pods -n production
kubectl top nodes

# Debug networking
kubectl debug node/NODE_NAME -it --image=ubuntu

# Check persistent volume status
kubectl get pvc -n production
kubectl describe pvc PVC_NAME -n production

# View events
kubectl get events -n production --sort-by='.lastTimestamp'
```

14.3 Best Practices Checklist

Development:

- ✅ Use TypeScript strict mode
- ✅ Add comprehensive unit tests
- ✅ Use dependency injection (NestJS)
- ✅ Validate all inputs
- ✅ Implement proper error handling

Docker:

- ✅ Use multi-stage builds
- ✅ Run as non-root user
- ✅ Include health checks
- ✅ Keep images small (<200MB)
- ✅ Scan for vulnerabilities

Kubernetes:

- ✅ Set resource requests/limits
- ✅ Use health checks (liveness/readiness)
- ✅ Implement PodDisruptionBudget
- ✅ Use NetworkPolicies
- ✅ Enable RBAC

Security:

- ✅ Use secrets for sensitive data
- ✅ Enable TLS/SSL everywhere
- ✅ Implement authentication/authorization

- ✅ Regular security audits
- ✅ Scan dependencies for CVEs

15. Production Checklist & Operations Guide

15.1 Pre-Production Deployment Checklist

Infrastructure

- ☐ EKS cluster created and tested
- ☐ RDS PostgreSQL backup verified
- ☐ S3 buckets configured with versioning
- ☐ VPC security groups configured
- ☐ IAM roles and policies set up
- ☐ SSL certificates provisioned

Application

- ☐ Unit tests passing
- ☐ Integration tests passing
- ☐ Code review completed
- ☐ Security scan passed
- ☐ Database migrations tested
- ☐ Environment variables documented

Deployment

- ☐ Docker images built and pushed to ECR
- ☐ Kubernetes manifests validated
- ☐ CI/CD pipeline tested
- ☐ ArgoCD configured
- ☐ Ingress and TLS working
- ☐ Health checks configured

Monitoring

- ☐ Prometheus metrics configured
- ☐ CloudWatch logs set up
- ☐ Grafana dashboards created
- ☐ Alarms configured
- ☐ On-call rotation established
- ☐ Runbooks documented

15.2 Operational Runbooks

Scenario 1: Deploying New Version

1. Push code to GitHub main branch
2. GitHub Actions builds Docker images
3. Images pushed to ECR
4. Manifest updated with new tag
5. ArgoCD syncs automatically
6. Rolling update deploys new pods
7. Monitor logs: `kubectl logs -f deployment/nestjs-api`

Scenario 2: Scaling Up for Traffic Spike

1. Monitor metrics in Grafana
2. HPA automatically scales up to 20 replicas
3. Or manually: `kubectl scale deployment nestjs-api --replicas=10`
4. Verify replicas: `kubectl get deployment nestjs-api`

Scenario 3: Database Connection Issues

1. Check RDS instance: `aws rds describe-db-instances`
2. Verify security groups allow traffic
3. Check connection pool: review logs for "too many connections"
4. Increase pool size in NestJS config
5. Restart pods: `kubectl rollout restart deployment/nestjs-api`

15.3 Maintenance Windows

Weekly Tasks:

- Review logs for errors
- Monitor cost trends
- Check security alerts

Monthly Tasks:

- Test backup/restore procedures
- Update dependencies
- Review and optimize performance
- Capacity planning review

Quarterly Tasks:

- Disaster recovery drill
- Security audit
- Database optimization
- Cost optimization review

15.4 Cost Optimization

Optimization	Savings	Effort
Reserved EC2 Instances (1 year)	30-40%	Low
Auto-scaling down off-peak	20-30%	Medium
Spot Instances for non-critical	60-70%	High
S3 Intelligent-Tiering	10-20%	Low
RDS Reserved Instances	30-50%	Low

15.5 Conclusion & Next Steps

You Now Have:

- ✅ Production-grade NestJS + Angular architecture
- ✅ Containerized application with Docker best practices
- ✅ Kubernetes orchestration on AWS EKS
- ✅ Scalable database with RDS PostgreSQL
- ✅ File storage with AWS S3

- ✅ CI/CD pipeline with GitHub Actions + ArgoCD
- ✅ Comprehensive monitoring and logging
- ✅ Security and compliance controls
- ✅ Operational runbooks and best practices

Next Steps:

1. Set up your GitHub repository with the code
2. Configure AWS account and services
3. Create Kubernetes cluster with EKS
4. Deploy application using provided manifests
5. Set up monitoring and alerting
6. Run load tests to validate performance
7. Execute pre-production checklist
8. Deploy to production
9. Continuously monitor and optimize

Support & Professional Services:

For implementation assistance, architecture reviews, or managed services:

📧 sales@bithost.in

🌐 www.bithost.in

Bithost - Cloud Infrastructure & Kubernetes Specialists

Enterprise Solutions | DevOps | Kubernetes Consulting | Managed Services

📧 sales@bithost.in | 🌐 www.bithost.in