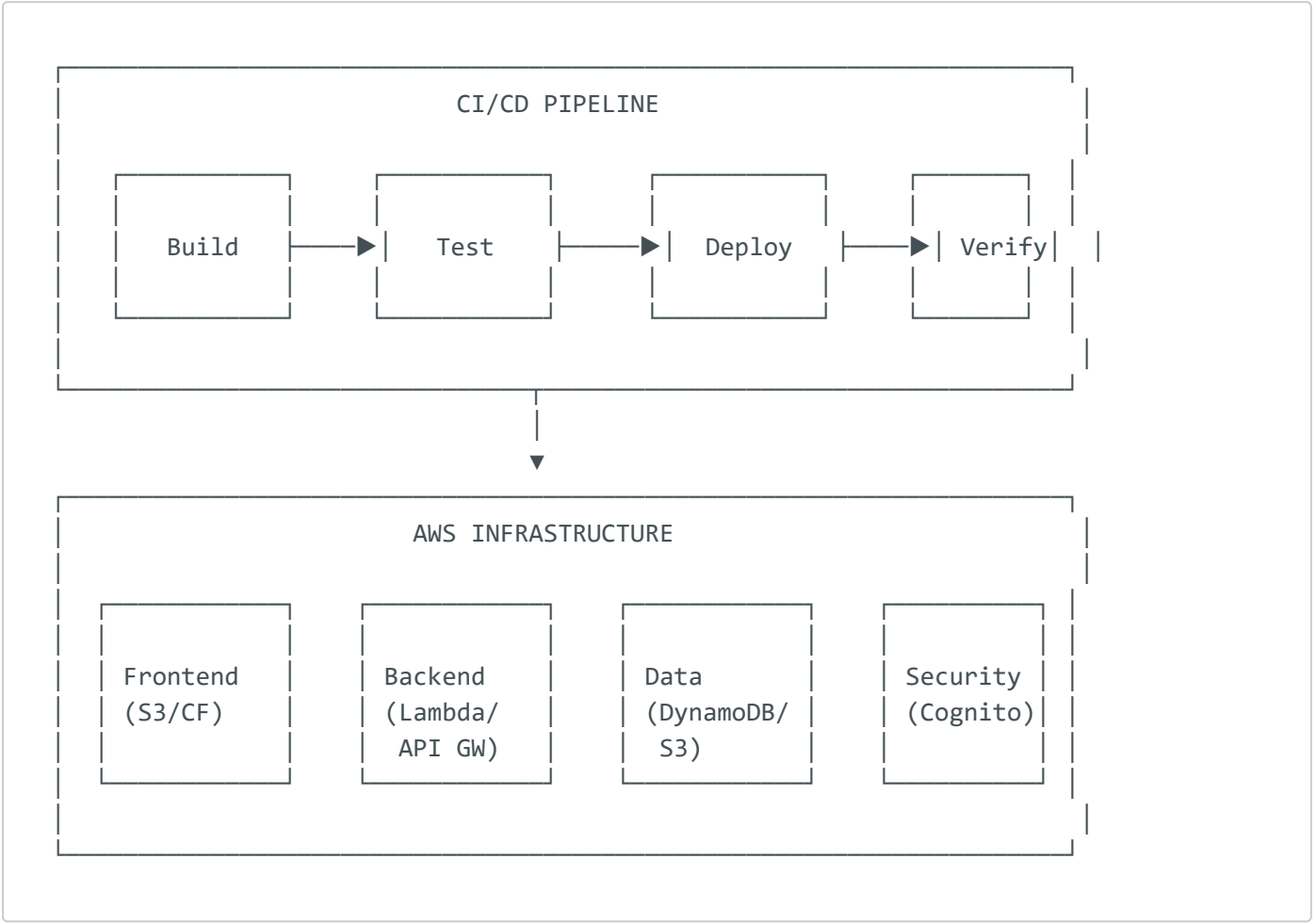
- Deployment Checklist

Deployment Architecture

This document outlines the deployment architecture for the Document Processing Accelerator, covering deployment strategies, environments, CI/CD pipelines, and operational considerations.

Deployment Overview

The Document Processing Accelerator follows a microservices deployment approach with infrastructure-as-code principles:



Deployment Components

Frontend Deployment

The React frontend is deployed to AWS S3 and CloudFront:

1. **Build Process**:
   - `npm run build` generates static assets
   - Assets are optimized for production
2. **Deployment**:
   - Static assets uploaded to S3 bucket
   - CloudFront distribution serves content with global edge caching
   - HTTPS enforced via CloudFront
3. **Deployment Steps**:

```
# Build the frontend
cd frontend
npm run build

# Deploy to S3
aws s3 sync build/ s3://document-processing-
accelerator-${ENVIRONMENT}-${SUFFIX} --delete

# Invalidate CloudFront cache
aws cloudfront create-invalidation --distribution-id ${CLOUDFRONT_ID} --paths
"/*"
```

Backend Deployment

The serverless backend is deployed using the Serverless Framework:

1. **Deployment Process**:
   - Infrastructure prerequisites deployed via Terraform
   - Lambda functions and API Gateway deployed via Serverless Framework
2. **Deployment Command**:

```
cd backend
npx serverless deploy --stage ${ENVIRONMENT}
```

3. **Generated Resources**:

- Lambda functions for each endpoint
- API Gateway with configured routes
- CloudWatch Logs for monitoring
- IAM roles and permissions

Infrastructure as Code

Terraform for Core Infrastructure

Terraform manages the core infrastructure components:

```
# Example Terraform configuration
module "frontend_s3" {
  source = "../../modules/s3"
  bucket_name = "document-processing-
accelerator-${var.environment}-${random_string.bucket_suffix.result}"
  environment = var.environment
}

module "frontend_cloudfront" {
  source = "../../modules/cloudfront"
  s3_bucket_name = module.frontend_s3.bucket_id
  s3_bucket_regional_domain_name = module.frontend_s3.bucket_regional_domain_name
  environment = var.environment
}

module "cognito" {
  source = "../../modules/cognito"
  environment = var.environment
  frontend_url = var.frontend_url
  documents_bucket_arn = module.documents_s3.bucket_arn
  api_gateway_arn = var.api_gateway_arn
}
```

Serverless Framework for Lambda and API Gateway

Serverless Framework manages the Lambda functions and API Gateway:

```
# Example serverless.yml excerpt
service: document-processing-api
```

```
  provider:
    name: aws
    runtime: nodejs18.x
    region: ${opt:region, 'us-east-1'}
    stage: ${opt:stage, 'dev'}
    environment:
      DOCUMENTS_TABLE: doc-processor-${self:provider.stage}-documents
      DOCUMENTS_BUCKET: doc-processor-${self:provider.stage}-
  documents-${env:BUCKET_SUFFIX, ''}
      OPENAI_API_KEY: ${env:OPENAI_API_KEY, ''}

  functions:
    getDocuments:
      handler: src/functions/documents/get.handler
      events:
        - http:
            path: /documents
            method: get
            cors: true
            authorizer:
              type: COGNITO_USER_POOLS
              authorizerId: !Ref ApiGatewayAuthorizer
```

Deployment Environments

The system supports multiple deployment environments:

Environment Strategy

Environment Configuration

Each environment has its own configuration:

Environment	Purpose	Scale	Security
Development	Feature development and testing	Minimal resources	Relaxed for testing

Environment	Purpose	Scale	Security
Staging	Pre-production validation	Production-like	Production-like
Production	Live system	Full scale	Strict security controls

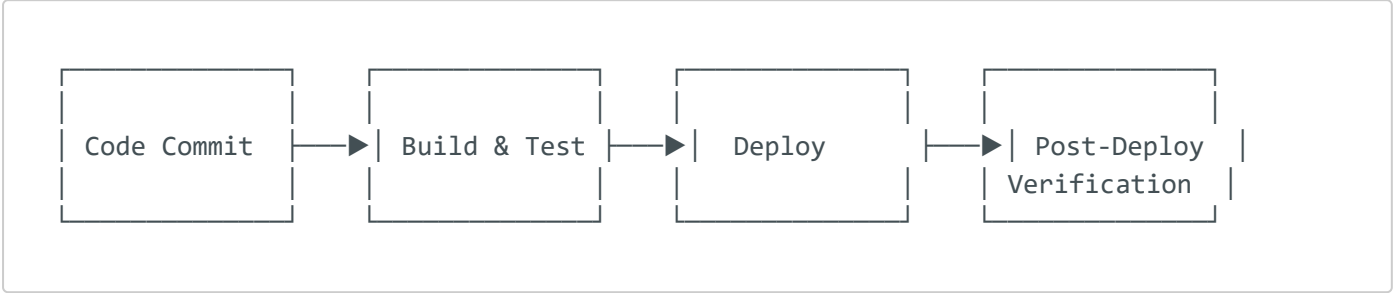
Environment Variables

Environment-specific variables are managed through:

- `.env.development` / `.env.production` for frontend
- Environment variables in CI/CD systems
- AWS Parameter Store for sensitive values

CI/CD Pipeline

The system implements a continuous integration and deployment pipeline:



Pipeline Components

1. **Code Commit Triggers**:
    - Pull request creation/update
    - Merge to development/staging/production branches
    - Tagged releases
2. **Build & Test Stage**:
    - Install dependencies
    - Run unit tests
    - Run integration tests

- Code quality checks

3. **Deployment Stage**:
   - Terraform infrastructure deployment
   - Backend deployment via Serverless Framework
   - Frontend build and deployment
4. **Verification Stage**:
   - Smoke tests
   - API validation
   - Security checks

CI/CD Implementation

The CI/CD pipeline can be implemented using:

- GitHub Actions
- AWS CodePipeline
- Jenkins

Example GitHub Actions workflow:

```
name: Deploy to Production

on:
  push:
    branches: [main]

jobs:
  deploy-infrastructure:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
      - name: Terraform Init
        run: cd terraform/environments/prod && terraform init
      - name: Terraform Apply
        run: cd terraform/environments/prod && terraform apply -auto-approve
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          TF_VAR_openai_api_key: ${{ secrets.OPENAI_API_KEY }}

  deploy-backend:
```

```
    needs: deploy-infrastructure
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '18'
      - name: Install dependencies
        run: cd backend && npm install
      - name: Deploy Backend
        run: cd backend && npx serverless deploy --stage prod
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          OPENAI_API_KEY: ${{ secrets.OPENAI_API_KEY }}
          BUCKET_SUFFIX: ${{ secrets.BUCKET_SUFFIX }}

  deploy-frontend:
    needs: deploy-backend
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '18'
      - name: Install dependencies
        run: cd frontend && npm install
      - name: Build
        run: cd frontend && npm run build
        env:
          REACT_APP_API_URL: ${{ secrets.PROD_API_URL }}
          REACT_APP_AWS_REGION: ${{ secrets.AWS_REGION }}
          REACT_APP_COGNITO_USER_POOL_ID: ${{ secrets.COGNITO_USER_POOL_ID }}
          REACT_APP_COGNITO_CLIENT_ID: ${{ secrets.COGNITO_CLIENT_ID }}
          REACT_APP_COGNITO_IDENTITY_POOL_ID: ${{ secrets.COGNITO_IDENTITY_POOL_ID
}}
      - name: Deploy to S3
        run: aws s3 sync frontend/build/ s3://${{ secrets.FRONTEND_BUCKET }} --
delete
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      - name: Invalidate CloudFront
        run: aws cloudfront create-invalidation --distribution-id ${{
secrets.CLOUDFRONT_ID }} --paths "/*"
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

Authentication Deployment

The Document Processing Accelerator uses a dual authentication system:

Local Development Authentication

- **Mock Authentication Service**: Implemented in `mockAuthService.ts`
- **Local Storage**: Authentication state stored in browser
- **No External Dependencies**: Works offline without AWS services

Production Authentication

- **AWS Cognito Integration**: Implemented in `amplifyAuthService.ts`
- **Amplify v5 API**: Modern authentication library
- **JWT Tokens**: Secure API authorization

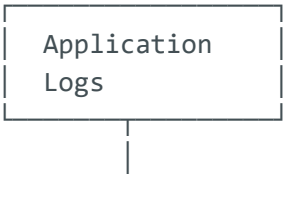
Authentication Configuration

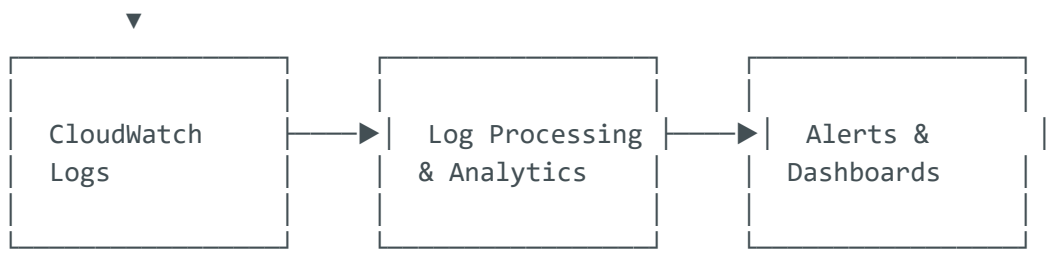
The authentication method is determined at build time:

```
// authServiceProvider.ts
import { AuthService } from '../types/auth';
import { mockAuthService } from './mockAuthService';
import { amplifyAuthService } from './amplifyAuthService';

// Determine which auth implementation to use
const useRealAuth = process.env.REACT_APP_USE_REAL_AUTH === 'true';

// Export the appropriate auth service implementation
export const authService: AuthService = useRealAuth
  ? amplifyAuthService
  : mockAuthService;
```

Operations and Monitoring

Logging Strategy





Monitoring Components

1. **Metrics Collection**:
    - Lambda execution metrics
    - API Gateway request metrics
    - CloudFront distribution metrics
    - Custom business metrics
2. **Alerting**:
    - Error rate thresholds
    - P95 latency thresholds
    - Lambda throttling alerts
    - API Gateway 4xx/5xx rate alerts
3. **Dashboards**:
    - Operational health
    - Business metrics
    - Cost optimization

Serverless Monitoring Tools

- CloudWatch Logs and Metrics
- X-Ray for distributed tracing
- CloudWatch Alarms for alerting
- CloudWatch Dashboards for visualization

Scaling Considerations

Frontend Scaling

- **CloudFront** handles scaling automatically at the edge
- **S3** scales infinitely for static content

Backend Scaling

- **Lambda** auto-scales based on concurrent requests
- **API Gateway** scales automatically to thousands of requests per second
- **DynamoDB** on-demand capacity mode for automatic scaling

Scaling Limits and Considerations

- Lambda concurrent execution limits (default: 1000)
- API Gateway rate limits (default: 10,000 RPS)
- DynamoDB throughput capacity
- S3 request rate limits (default: 5,500 GET/s per prefix)

Disaster Recovery

Backup Strategy

- **DynamoDB**: Point-in-time recovery enabled
- **S3**: Versioning and cross-region replication
- **Configuration**: Infrastructure as Code for quick recovery

Recovery Time Objectives (RTO)

Component	RTO	Recovery Method
Frontend	< 30 minutes	Redeploy from source to standby region
Backend	< 60 minutes	Redeploy Lambda/API to standby region
Database	< 30 minutes	DynamoDB global tables / point-in-time

Cost Optimization

Cost-Saving Strategies

1. **Lambda Optimization**:
    - Right-sized memory allocation
    - Cold start optimization
    - Code bundling optimization
2. **API Gateway Optimization**:
    - Response caching
    - Request validation to prevent unnecessary Lambda invocations
3. **S3 & CloudFront Optimization**:
    - Appropriate content caching
    - Compression
    - Intelligent routing
4. **OpenAI API Optimization**:
    - Prompt engineering for token efficiency
    - Model selection based on task complexity
    - Result caching

Cost Monitoring

- CloudWatch Billing Alarms
- AWS Cost Explorer tracking
- Cost allocation tags

Security Operations

Security Monitoring

1. **Authentication Monitoring**:
    - Failed login attempts
    - Suspicious login patterns
    - Token usage analytics
2. **API Security Monitoring**:
    - Rate limiting and throttling
    - Input validation failures
    - Authorization failures
3. **Data Access Monitoring**:
    - S3 access logs
    - DynamoDB access patterns
    - Cross-origin resource sharing (CORS) violations

Compliance Monitoring

- Automated compliance checks
- Regular security audits
- Vulnerability scanning

Frontend Deployment Guide

A step-by-step guide for developers:

1. **Local Development Setup**:

```
cd frontend
npm install
npm start
```

2. **Environment Variables Configuration**: Create a `.env.local` file with required variables or set them in the CI/CD system.
3. **Production Build Process**:

```
npm run build
```

4. **AWS Deployment Steps**:

```
aws s3 sync build/ s3://document-processing-accelerator-dev-[suffix] --delete
aws cloudfront create-invalidation --distribution-id DISTRIBUTION_ID --paths
"/*"
```

5. **Verification Steps**:
    - Verify that the application loads in the browser
    - Check that authentication works
    - Test main functionality
    - Verify API connectivity

Deployment Checklist

Before deploying to production:

1. **Code Quality**:
    - All tests passing
    - Code reviews completed
    - Static analysis passed
2. **Security**:
    - Authentication and authorization verified
    - Sensitive information properly secured
    - API endpoints properly protected
3. **Performance**:
    - Frontend bundles optimized
    - API response times acceptable
    - Database queries optimized
4. **Documentation**:

- API documentation updated
- Deployment documentation current
- User documentation ready