Deployment Architecture

> **Source File:** generated-documents\technical-design\deployment-architecture.md
> **Generated:** 15/07/2025 at 11:40:55
> **Generated by:** Requirements Gathering Agent - PDF Converter

DeploymentArchitecture

Generated by adpa-enterprise-framework-automation v3.2.0
Category: technical-design
Generated: 2025-07-14T21:08:17.896Z
Description:

ADPA (Advanced Document Processing & Automation Framework)

Deployment Architecture Document

---

1. Deployment Overview

ADPA is a modular, enterprise-grade automation framework for AI-powered document generation, project management, and business analysis. Built with Node.js and TypeScript, it provides a CLI, REST API

(Express.js), and a Next.js admin portal, with integrations for AI providers (OpenAI, Google AI, Copilot, Ollama), enterprise document management (Confluence, SharePoint, Adobe), and compliance with standards (BABOK, PMBOK, DMBOK).

ADPA is designed for secure, scalable, multi-environment deployments (dev, staging, production), supporting horizontal scaling, HA, and enterprise security best practices. The deployment workflow is container- and cloud-ready, with extensibility for Kubernetes orchestration and CI/CD automation.

---

2. Infrastructure Architecture

2.1 Logical Architecture

```
 +-----------------+     +-----------------+
|   End Users     | --> |  Load Balancer   | <---+
+-----------------+     +-----------------+     |
          |                                  +---+---+
          v                                  v       v
+-----------------+   +-----------------+  +-----------------+
|  API Server(s)  |   |  Admin Portal   |  |   CLI Clients   |
| (Express.js)    |   | (Next.js)       |  +-----------------+
+-----------------+   +-----------------+           |
        |                      |                    |
        v                      v                    v
+-------------------------------------------------------------+
|                   Integration Layer                         |
|  (Confluence, SharePoint, Adobe, VCS, AI Providers)         |
+-------------------------------------------------------------+
        |
        v
+-------------------+
|  Data Storage     | (JSON files, extensible to SQL/NoSQL)
+-------------------+
```

2.2 Physical Architecture

- **API Server**: Node.js/Express.js, containerized (Docker), horizontally scalable.
- **Admin Portal**: Next.js (React), containerized, separate deployment.
- **CLI**: Node.js binary or global npm install, used by authenticated users.
- **Load Balancer**: Azure Application Gateway, AWS ALB, or NGINX.
- **Integration Endpoints**: VPC egress for external APIs (OpenAI, Google AI, SharePoint, Confluence, Adobe).
- **Persistent Storage**: Cloud file storage (Azure Files, AWS EFS) or managed DB (Azure SQL, CosmosDB, MongoDB Atlas, etc.).
- **Cache (optional)**: Redis for performance and session management.

---

3. Environment Setup

3.1 Environments

- **Development**: Local, Docker Compose, or cloud dev environment.
- **Staging**: Mirrors production; used for QA and integration testing.
- **Production**: HA, secure, with monitoring, backup, and compliance.

3.2 Prerequisites

- **Node.js**: >=18.0.0
- **npm**: >=9.x
- **Docker**: (for containerized deployments)
- **Cloud Provider**: Azure, AWS, GCP, or on-premises VM
- **External API Keys**: OpenAI, Google AI, Azure OpenAI, Adobe, Microsoft Graph, etc.

3.3 Directory Structure

```
requirements-gathering-agent/
├── src/                    # Core services (API, CLI, modules)
├── admin-interface/       # Next.js admin portal
├── api-specs/             # OpenAPI/TypeSpec specs
├── test/                  # Unit/integration tests
├── docs/                  # Documentation
├── generated-documents/   # Generated outputs
├── dist/                  # Compiled JS
├── .env                   # Environment variables
```

---

4. Deployment Process

4.1 Build and Package

CLI/NPM:

```
npm install -g adpa-enterprise-framework-automation
```

From Source:

```
git clone https://github.com/mdresch/requirements-gathering-agent.git
cd requirements-gathering-agent
npm install
npm run build
```

Docker (when available):

```
docker pull adpa/enterprise-framework:latest
# or build locally
docker build -t adpa/enterprise-framework:latest .
```

4.2 Deploying Components

API Server (Express.js)

- Run as a container or Node.js process (preferably behind a load balancer)
- Expose port (default: 3000/3001)
- Use `.env` for configuration

Admin Portal (Next.js)

- Deploy as a container, static site, or managed service (Vercel/Azure Static Web Apps)
- Expose on a separate port (default: 3001)
- Configure API endpoint in environment

CLI

- Distributed via npm or as a binary; requires network access to API Server

4.3 Integration Endpoints

- **Confluence/SharePoint/Adobe**: Requires OAuth2 setup and secure storage of credentials.
- **AI Providers**: API keys/secrets in environment variables or mounted secrets.

4.4 Continuous Deployment (Recommended)

- Use CI/CD pipelines (GitHub Actions, Azure DevOps, GitLab CI)
- Steps: Lint/Test → Build → Containerize → Deploy to Staging → Approve → Deploy to Production

---

5. Configuration Management

5.1 Environment Variables

- `.env` file for each environment (`.env` , `.env.production` , `.env.staging` , `.env.local`)

- Use secret management (Azure Key Vault, AWS Secrets Manager) for sensitive keys in production

Sample `.env`:

```
NODE_ENV=production
PORT=3001
OPENAI_API_KEY=xxxx
GOOGLE_AI_API_KEY=xxxx
AZURE_OPENAI_ENDPOINT=xxxx
CONFLUENCE_CLIENT_ID=xxxx
SHAREPOINT_CLIENT_ID=xxxx
JWT_SECRET=xxxx
ALLOWED_ORIGINS=https://yourdomain.com
```

5.2 Configuration Best Practices

- Never commit secrets to source control
- Use `.env.example` as a template
- Support runtime configuration reload (where possible)
- Validate config at startup (with Joi/Zod)

---

6. Scaling Strategy

6.1 API Server

- **Horizontal Scaling**: Deploy multiple stateless API server instances behind a load balancer.
- **Stateless Design**: Store session data in Redis or JWT, persistent data in cloud storage/DB.
- **Auto-Scaling**: Use Kubernetes HPA, Azure App Service autoscale, or AWS ECS/Fargate scaling policies.

6.2 Admin Portal

- Scaled independently from API server.
- Use CDN for static assets.

6.3 Integration Layer

- Implement connection pooling and retry logic for third-party APIs.
- Use circuit breaker patterns for rate-limited or unreliable endpoints.

6.4 Caching

- Use Redis for caching frequent queries, authentication tokens, and session data.

---

7. Monitoring Setup

7.1 Logging

- Use Winston/Morgan for server logs.
- Centralize logs to ELK Stack, Azure Monitor, or AWS CloudWatch.
- Log authentication events, errors, API usage, and integration failures.

7.2 Metrics and Health Checks

- Expose `/api/v1/health` and `/api/v1/health/ready` endpoints.
- Integrate with Prometheus/Grafana for metrics (requests/sec, error rates, latency).
- Use Application Insights (Azure) or CloudWatch (AWS) for distributed tracing.

7.3 Alerting

- Set up alerts for failed health checks, high error rates, and integration timeouts.

---

8. Backup and Recovery

8.1 Data Backup

- **Configuration/JSON/Document Output**: Back up generated documents and configuration files to cloud storage (Azure Blob, AWS S3) on a scheduled basis.
- **Database**: If using SQL/NoSQL, use platform-native backup features (automated snapshots, point-in-time restore).

8.2 Restoration

- Document restore procedures (download from backup, restore to correct location).
- Regularly test restore processes.

---

9. Disaster Recovery

9.1 Redundancy

- Deploy across multiple availability zones/regions.
- Use managed cloud services where possible.

9.2 Failover

- Configure load balancers for health-based routing.
- Maintain hot/cold standby for critical components.

9.3 DR Testing

- Schedule DR drills.
- Maintain up-to-date runbooks for system restore and failover.

---

10. Maintenance Procedures

10.1 Routine Maintenance

- **Dependency Updates**: Regularly update npm packages and patch vulnerabilities (use Dependabot, npm audit).
- **Key Rotation**: Rotate API keys/secrets on a schedule.
- **Configuration Review**: Audit environment variables and secret storage.
- **Resource Monitoring**: Monitor CPU/memory usage, scale resources as needed.

10.2 Security Maintenance

- Review audit logs for suspicious activity.
- Apply OS/container security patches.
- Review third-party integration permissions/scopes.

10.3 Documentation

- Keep architecture and runbooks up-to-date.
- Document configuration changes and deployment history.

---

11. Security Considerations

- **Authentication**: JWT, OAuth2, SAML, and Active Directory integration supported.
- **Authorization**: Role-based access control (RBAC) for API and admin portal.
- **Transport Security**: Enforce HTTPS/TLS for all endpoints.
- **Input Validation**: Use Joi/Zod for API payload validation.
- **Rate Limiting**: Prevent abuse (express-rate-limit).
- **CORS**: Restrict origins for API access.
- **Secrets Management**: Use vaults, never hardcode secrets.

- **Compliance**: GDPR, SOX, PCI DSS, HIPAA, and other applicable standards.

---

12. Dependencies

Production Dependencies:

- Node.js >=18, TypeScript >=5.7, Express.js, Next.js, React
- Integration SDKs: @adobe/pdfservices-node-sdk, @azure/msal-node, @azure/openai, @google/generative-ai, @microsoft/microsoft-graph-client, openai, yargs, winston, etc.
- Security: bcryptjs, express-rate-limit, helmet, cors, dotenv, jsonwebtoken
- Validation: joi, zod
- Logging: morgan, winston

Dev Dependencies:

- Jest, ts-jest, ESLint, Prettier, TypeSpec, Swagger

---

13. Deployment Workflow Summary

1. **Prepare Environment**: Set up cloud infra, storage, secrets.
2. **Configure Environment Variables**: Fill out `.env` or use secret manager.
3. **Build Artifacts**: `npm run build` or Docker image build.
4. **Deploy API Server**: As container or Node.js process, behind a load balancer.
5. **Deploy Admin Portal**: As container/static site; configure API endpoint.
6. **Integrate Third-Party Services**: Register and secure credentials.
7. **Apply Security Policies**: Set up RBAC, CORS, HTTPS.
8. **Run Health Checks**: Verify `/health` and `/health/ready` endpoints.

9. **Monitor & Test**: Monitor logs, performance, and run automated tests.
10. **Backup Data**: Schedule regular backups.
11. **Document and Review**: Update docs, review logs, plan for scaling or DR as needed.

For detailed step-by-step cloud deployment templates (Docker Compose, Kubernetes, Azure/AWS scripts), see project documentation or contact DevOps support.
