

# TECHNICAL ARCHITECTURE & DEVOPS OPERATIONS



## GEOCODE OVERVIEW

---

<b>Platform</b>	Microsoft Azure Cloud
<b>Frontend</b>	Nuxt / Vue.js
<b>Backend API</b>	Spring Boot (Java)
<b>Database</b>	Azure Database for MySQL / MS DB
<b>Security</b>	Auth0 Identity Platform
<b>CI/CD</b>	Azure DevOps Pipelines
<b>Hosting</b>	Azure App Services

Document Version 1.0 | February 12, 2026

# Table of Contents

---

- Table of Contents ..... 2
- 1. Executive Summary ..... 3
- 2. Application Architecture ..... 3
  - 2.1 Architecture Overview ..... 3
  - 2.2 Frontend — Nuxt / Vue.js ..... 4
  - 2.3 Backend API — Spring Boot..... 4
  - 2.4 Database — Azure Database for MySQL ..... 4
- 3. Security Architecture ..... 5
  - 3.1 Authentication & Authorization — Auth0 ..... 5
  - 3.2 Transport & Network Security ..... 5
  - 3.3 Security Responsibility Matrix..... 5
- 4. Environments ..... 6
- 5. Source Control & Branching Strategy..... 6
  - 5.1 Branch Structure ..... 6
  - 5.2 Code Promotion Flow ..... 7
  - 5.3 Branch Protection & Governance ..... 7
- 6. CI/CD Pipeline & Deployment ..... 7
  - 6.1 Automated Pipeline Stages ..... 8
  - 6.2 Deployment Environments in Azure DevOps ..... 8
  - 6.3 Application Configuration Management ..... 8
- 7. Azure Infrastructure ..... 9
  - 7.1 Azure Resource Summary..... 9
  - 7.2 Azure App Service Benefits ..... 9
- 8. Operational Procedures ..... 10
  - 8.1 Standard Release Process ..... 10
  - 8.2 Incident Response — Production Issue ..... 10
  - 8.3 Monitoring & Observability ..... 11
- 9. Roles & Responsibilities..... 11
- 10. Technology Stack Reference ..... 12
- 11. Glossary..... 12

# 1. Executive Summary

This document provides a comprehensive overview of the PESC GeoCode application platform, its underlying technology stack, cloud infrastructure, security posture, and software development lifecycle. The platform is built on industry-standard, enterprise-grade technologies hosted entirely within the Microsoft Azure cloud ecosystem, ensuring scalability, reliability, and maintainability.

**Platform Summary**

The application is a full-stack web platform consisting of a modern Vue/Nuxt frontend, a Spring Boot REST API backend, and a managed Azure relational database. All services run on Azure App Services with Auth0 providing enterprise-grade identity and access management. Development, testing, and deployment are fully automated through Azure DevOps CI/CD pipelines.

The architecture is designed to support two isolated environments — Test and Production — enabling teams to validate all changes in a staging context before promoting code to end users. This approach minimizes risk, reduces downtime, and ensures production stability.

## 2. Application Architecture

The application follows a modern three-tier architecture pattern: a decoupled frontend that communicates with a RESTful backend API, which in turn reads and writes from a managed cloud database. Each tier is independently deployed and scaled on Azure App Services.

### 2.1 Architecture Overview

Tier	Technology	Description
Presentation	Nuxt / Vue.js	Server-side rendered and static SPA frontend, served as a Node.js application on Azure App Service
Application	Spring Boot (Java)	RESTful API layer handling all business logic, data validation, and integration with Auth0 and the database tier
Data	Azure MS DB (MySQL)	Fully managed, highly available relational database service hosted in Azure with automated backups and geo-redundancy

Tier	Technology	Description
Identity / Auth	Auth0	External identity provider managing user authentication, OAuth 2.0 / OIDC token issuance, and role-based access control (RBAC)
Hosting	Azure App Services	PaaS hosting environment for both frontend and backend applications, providing auto-scaling, health monitoring, and managed TLS

## 2.2 Frontend — Nuxt / Vue.js

The user interface is built with Vue.js, a progressive JavaScript framework, and wrapped in the Nuxt framework which provides server-side rendering (SSR), file-based routing, and production-grade performance optimizations. The frontend is deployed as a Node.js application on Azure App Service.

- Server-side rendering (SSR) for improved SEO and initial page load performance
- Component-based architecture enabling reusable UI elements and consistent design
- Communicates with the Spring Boot backend exclusively over HTTPS REST APIs
- Auth0 SDK integration for seamless user login, logout, and session management
- Deployed as an Azure App Service (Linux) with automatic restarts and health probes

## 2.3 Backend API — Spring Boot

The backend is a Java-based Spring Boot application that exposes a RESTful API consumed by the frontend. It encapsulates all business logic, enforces data validation rules, and manages communication with the Azure database and Auth0.

- RESTful API design following HTTP standards (GET, POST, PUT, DELETE)
- Spring Security integration with Auth0 JWT token validation on all protected endpoints
- JPA / Hibernate ORM layer for type-safe, maintainable database interaction
- Environment-specific configuration using Azure App Service Application Settings
- Deployed as an Azure App Service (Linux) with automated build and restart on code push

## 2.4 Database — Azure Database for MySQL

Data is persisted in Azure Database for MySQL (MS DB), a fully managed relational database service. Microsoft manages infrastructure maintenance, patching, backups, and high availability, reducing operational overhead significantly.

- Fully managed PaaS database with automatic patching and version management
- Built-in high availability with automatic failover (99.99% SLA)
- Automated daily backups with point-in-time restore capability
- Data encryption at rest and in transit enforced by default
- Access restricted to Azure App Service via private VNet integration or firewall rules

### 3. Security Architecture

Security is addressed at multiple layers of the stack — from identity and access management to transport-layer encryption and cloud network controls. Auth0 serves as the centralized identity provider, eliminating the need to manage password storage and authentication infrastructure internally.

#### 3.1 Authentication & Authorization — Auth0

Auth0 is an enterprise-grade Identity-as-a-Service (IDaaS) platform that handles the full authentication lifecycle. It supports industry-standard protocols including OAuth 2.0 and OpenID Connect (OIDC), ensuring compatibility and compliance with modern security standards.

Capability	Implementation
Authentication Protocol	OAuth 2.0 / OpenID Connect (OIDC)
Token Format	JSON Web Tokens (JWT) — signed and validated by Spring Boot API
User Management	Auth0-hosted user directory with social login and enterprise SSO options
Role-Based Access Control	Auth0 RBAC with roles and permissions propagated via JWT claims
Multi-Factor Authentication	Available and configurable via Auth0 dashboard (SMS, TOTP, push)
Session Management	Auth0-managed sessions with configurable token expiry and refresh

#### 3.2 Transport & Network Security

- All traffic between users and the application is encrypted using TLS 1.2+ (HTTPS enforced by Azure App Service)
- Azure App Service managed TLS certificates — no manual certificate provisioning required
- Database access restricted to whitelisted Azure App Service IP ranges or private endpoint
- Auth0 tokens are validated on every API request — no unauthenticated endpoints exposed
- Environment secrets (database credentials, Auth0 secrets) stored in Azure App Service Application Settings, never in source code

#### 3.3 Security Responsibility Matrix

Security Domain	Owner	Tooling
User Authentication	Auth0	Auth0 Universal Login
API Authorization	Spring Boot + Auth0	JWT / Spring Security

Security Domain	Owner	Tooling
Data Encryption at Rest	Microsoft Azure	Azure Managed Encryption
Data Encryption in Transit	Azure App Service	TLS 1.2+ Managed Cert
Secret & Config Management	Dev Team / Azure	App Service App Settings
DB Access Control	Azure Networking	Firewall / Private Endpoint
OS & Platform Patching	Microsoft Azure	Managed PaaS (automatic)

## 4. Environments

The platform operates across two dedicated, isolated environments — Test and Production. This separation ensures that all code changes are validated in a realistic environment before being promoted to end users, dramatically reducing the risk of production incidents.

Environment	Git Branch	Purpose	Deployment Trigger
Test	Test-Main	Integration testing, QA validation, and stakeholder review	Automatic — on developer merge to Test-Main
Production	Main	Live end-user environment — stable, validated releases only	Automatic — on merge from Test-Main to Main

Each environment runs its own isolated instance of the frontend App Service, backend App Service, and is configured with environment-specific database connection strings and Auth0 tenant/application credentials via Azure App Service Application Settings.

## 5. Source Control & Branching Strategy

All source code is managed in Azure DevOps Repos. The branching model is designed to protect production stability while enabling parallel developer workstreams. It establishes a clear promotion path from individual development work through to production release.

### 5.1 Branch Structure

Branch	Type	Purpose & Policy
main	Protected	Production branch — represents the current live codebase. Direct commits are prohibited. Only accepts merges from Test-Main after successful QA sign-off.
test-main	Integration	Staging and integration branch — all developer branches are merged here. Triggers automatic build and deployment to the Test environment.
feature/*, bugfix/*, dev/*	Developer	Individual developer branches — short-lived branches created from Test-Main for specific features or bug fixes. Merged back to Test-Main when ready for review.

## 5.2 Code Promotion Flow

The following is the standard path a code change takes from a developer's workstation to the production environment:

### Code Promotion Workflow

- Step 1 — Developer creates a personal branch from Test-Main (e.g. feature/my-feature)
- Step 2 — Developer commits and pushes changes to their personal branch
- Step 3 — Developer opens a Pull Request (PR) from their branch into Test-Main
- Step 4 — PR is reviewed and approved, then merged into Test-Main
- Step 5 — Azure DevOps pipeline auto-triggers: builds application and deploys to Test environment
- Step 6 — Application automatically restarts in Test; QA and stakeholders validate
- Step 7 — Once testing is approved, Test-Main is merged into Main
- Step 8 — Azure DevOps pipeline auto-triggers: builds and deploys to Production environment

## 5.3 Branch Protection & Governance

- The Main branch is protected — direct pushes are not permitted under any circumstances
- All changes to Test-Main are introduced via Pull Request to enforce peer code review
- Production deployments are triggered exclusively by merges into the Main branch
- Developer branches are short-lived and should be deleted after successful merge
- Branch naming conventions (feature/, bugfix/, hotfix/) provide clear context for code review

## 6. CI/CD Pipeline & Deployment

Continuous Integration and Continuous Deployment (CI/CD) is implemented entirely within Azure DevOps Pipelines. The pipeline automates the build, test, and deployment process for both the Test and Production environments, ensuring consistency and eliminating manual deployment errors.

## 6.1 Automated Pipeline Stages

Stage	Name	Trigger	Actions Performed
1	Source Checkout	Branch push / PR merge	Pipeline clones the target branch from Azure DevOps Repos
2	Build — Frontend	Automatic	npm install, npm run build — compiles Nuxt/Vue.js application into production assets
3	Build — Backend	Automatic	mvn package (Maven build) — compiles Spring Boot Java application into a deployable JAR artifact
4	Deploy to Target Env	Automatic	Azure CLI / App Service deployment — pushes build artifacts to the appropriate App Service (Test or Production)
5	App Restart	Automatic	Azure App Service restart triggered to apply new deployment and reload environment configuration

## 6.2 Deployment Environments in Azure DevOps

Azure DevOps Environments are configured as deployment targets within the pipeline, providing deployment history, audit logging, and the ability to add manual approval gates. The Production environment can be configured to require explicit sign-off before deployment proceeds.

- Test Environment — deploys automatically with no manual gate, enabling fast feedback loops
- Production Environment — may be configured with a required approver gate for additional governance
- Deployment history is preserved in Azure DevOps for audit and rollback reference
- Failed deployments generate notifications and can trigger automatic rollback policies

## 6.3 Application Configuration Management

Sensitive configuration values — including database connection strings, Auth0 client secrets, and API keys — are stored exclusively as Azure App Service Application Settings. These values are injected as environment variables at runtime and are never committed to source control.

Configuration Item	Test Environment	Production Environment
Database Connection String	Test Azure MS DB instance	Production Azure MS DB instance
Auth0 Domain & Client ID	Test Auth0 Application	Production Auth0 Application
Auth0 Client Secret	Stored in App Settings (Test)	Stored in App Settings (Prod)
API Base URL	Test App Service URL	Production App Service URL
Log Level / Debug Flags	Verbose (DEBUG)	Minimal (WARN / ERROR)

## 7. Azure Infrastructure

All application infrastructure runs on Microsoft Azure using Platform-as-a-Service (PaaS) offerings. This approach eliminates the need for the organization to manage virtual machines, operating systems, or infrastructure patching — Microsoft handles all underlying platform operations.

### 7.1 Azure Resource Summary

Azure Resource	Service Type	Role in Platform
App Service — Frontend (Test)	Azure App Service (Linux)	Hosts the Nuxt/Vue.js Node.js application for the Test environment
App Service — Backend (Test)	Azure App Service (Linux)	Hosts the Spring Boot Java API JAR for the Test environment
App Service — Frontend (Prod)	Azure App Service (Linux)	Hosts the production Nuxt/Vue.js Node.js application
App Service — Backend (Prod)	Azure App Service (Linux)	Hosts the production Spring Boot Java API JAR
Azure MS DB (Test)	Azure Database for MySQL	Managed relational database for the Test environment
Azure MS DB (Production)	Azure Database for MySQL	Managed relational database for the Production environment
Azure DevOps Organization	Azure DevOps	Source control, CI/CD pipelines, project boards, and artifact storage
Auth0 Tenant	External SaaS (Auth0)	Identity provider for authentication and authorization across all environments

### 7.2 Azure App Service Benefits

Azure App Service is a fully managed PaaS offering that provides significant operational advantages over self-managed infrastructure:

- Automatic OS patching and security updates — no server maintenance required
- Built-in load balancing and auto-scaling to handle variable traffic demands
- Integrated TLS certificate management with automatic renewal
- Built-in health monitoring and automated restarts on failure
- Deployment slots available for zero-downtime blue/green deployments
- 99.95% uptime SLA for Standard and Premium tier App Service Plans

## 8. Operational Procedures

---

### 8.1 Standard Release Process

#### Standard Release Checklist

Pre-Release:

1. All developer branches merged into Test-Main and pipeline successfully completed
2. Test environment validated by QA — functional testing, regression testing, and UAT complete
3. Stakeholder sign-off obtained on Test environment
4. Database migration scripts reviewed and tested against Test database

Release Execution:

5. Pull Request opened from Test-Main to Main with release notes
6. PR reviewed and approved by designated release approver
7. Merge into Main triggers automated Production pipeline
8. Production deployment and app restart confirmed in Azure DevOps

Post-Release:

9. Production application health check verified
10. Critical user flows tested in production
11. Monitoring dashboards observed for 15-30 minutes post-deployment

### 8.2 Incident Response — Production Issue

- If a critical defect is detected post-release, a hotfix branch is created directly from Main
- The fix is developed, tested in isolation, and merged back to Main via PR for immediate deployment

- The hotfix is simultaneously back-merged into Test-Main to keep environments synchronized
- All incidents are documented and reviewed in a post-mortem to prevent recurrence

### 8.3 Monitoring & Observability

Azure App Service provides built-in diagnostics and logging. The following monitoring capabilities are available and should be configured:

Capability	Azure Service	Purpose
Application Logging	App Service Logs	Captures stdout/stderr from frontend and backend applications
HTTP Request Logging	App Service HTTP Logs	Records all inbound HTTP requests including status codes and latency
Application Performance	Azure Application Insights	End-to-end telemetry, dependency tracking, and performance profiling
Database Monitoring	Azure Monitor (MySQL)	Query performance, connection counts, and resource utilization metrics
Alerting	Azure Monitor Alerts	Configurable alerts for HTTP 5xx errors, CPU thresholds, and availability
Auth Anomalies	Auth0 Logs & Streams	Failed login attempts, suspicious activity, and token anomalies

## 9. Roles & Responsibilities

Role	Access Level	Responsibilities
Software Developer	Personal branch + Test-Main (write)	Develop features/fixes on personal branches, submit PRs to Test-Main, participate in code review
Tech Lead / Senior Dev	Test-Main + Main (write, gated)	Review and approve PRs, own Test-Main to Main promotion, maintain code quality standards
QA Engineer	Test environment (read + test)	Execute functional, regression, and UAT test plans in the Test environment; provide go/no-go sign-off
DevOps Engineer	Azure DevOps + Azure Portal	Maintain CI/CD pipelines, manage Azure infrastructure, configure App Settings, monitor health
Release Manager	Main branch (approve)	Approve production PRs, coordinate release scheduling, own post-release verification

Role	Access Level	Responsibilities
Security / IAM Admin	Auth0 Dashboard + Azure IAM	Manage Auth0 tenant configuration, user roles, application registrations, and access policies

## 10. Technology Stack Reference

Component	Technology	Version / Tier	Notes
Frontend Framework	Vue.js	Vue 3.x	Reactive component framework
Frontend Meta-Framework	Nuxt	Nuxt 3.x	SSR, routing, module ecosystem
Backend Framework	Spring Boot	3.x (Java 17+)	Auto-configured Spring MVC, Security, JPA
Database	Azure DB for MySQL	Flexible Server	Managed, HA, auto-backup
Identity Provider	Auth0	Enterprise / B2C	OAuth 2.0, OIDC, RBAC
Source Control	Azure DevOps Repos	Git	Branching + PR workflow
CI/CD	Azure DevOps Pipelines	YAML Pipelines	Build + Deploy automation
Hosting Platform	Azure App Services	Standard / Premium S-tier	PaaS, auto-scale, TLS mgmt
Build Tool — Backend	Apache Maven	3.x	Java dependency management and build
Build Tool — Frontend	Node.js / npm	Node 18+ LTS	Package management and build scripts

## 11. Glossary

Term	Definition
App Service	Microsoft Azure's Platform-as-a-Service (PaaS) web hosting environment. Abstracts all server management, patching, and scaling from the application team.

Term	Definition
Auth0	A cloud-hosted Identity-as-a-Service platform providing authentication, authorization, and user management via OAuth 2.0 and OIDC standards.
Azure DevOps	Microsoft's integrated DevOps platform encompassing source control (Repos), CI/CD (Pipelines), project management (Boards), and artifact management.
Branch (Git)	An isolated copy of the codebase used to develop a feature or fix without impacting other branches.
CI/CD	Continuous Integration / Continuous Deployment — the practice of automatically building, testing, and deploying code changes through an automated pipeline.
JWT	JSON Web Token — a compact, digitally signed token used to securely transmit identity and authorization claims between Auth0, the frontend, and the backend API.
Main Branch	The protected Git branch representing the current production codebase.
Nuxt	A meta-framework built on top of Vue.js that adds server-side rendering, file-based routing, and a module ecosystem for production-grade applications.
OAuth 2.0 / OIDC	Industry-standard protocols for delegated authorization (OAuth 2.0) and identity verification (OpenID Connect) used by Auth0 for secure user authentication.
PaaS	Platform-as-a-Service — a cloud service model where the provider manages infrastructure, OS, and runtime, allowing teams to focus exclusively on application code.
Pipeline	An automated Azure DevOps workflow that builds, tests, and deploys application code upon a trigger event (such as a branch merge).
Pull Request (PR)	A formal request to merge code from one branch into another, enabling peer code review before integration.
Spring Boot	An opinionated Java framework that simplifies the creation of production-ready REST APIs and microservices with minimal configuration.
Test-Main Branch	The integration Git branch that mirrors the Test environment. Developer branches are merged here to trigger automated deployment to the Test App Services.

**END OF DOCUMENT**

*This document is confidential and intended for internal use only. Do not distribute externally without authorization.*