ANDERSEN

Architecture Vision Document for ProjectY

Issued: 2025-01-15

Version: 1.1

Andersen Sp.z oo., rondo Daszyńskiego 1, 23th floor 00-843 Warsaw, Poland

Revision History

Date	Version	Description	Author
2024-12-10	0.1	Initial presentation of the vision	Dzehtsiarou Yauhen
2024-12-15	0.2	Executive Summary	Dzehtsiarou Yauhen
2024-12-23	0.3	Architectural drivers part is added	Dzehtsiarou Yauhen
2024-12-29	0.4	Solution Architecture	Dzehtsiarou Yauhen
2025-01-05	0.5	Operation plan	Dzehtsiarou Yauhen
2025-01-10	1.0	Implementation Roadmap	Dzehtsiarou Yauhen
2025-01-15	1.1	Adding details about architectural solutions, and container diagram	Dzehtsiarou Yauhen

1 Introduction

1.1 Vision Purpose

This document provides a high-level description of the technical solution for ProjectY. This solution was conceived with the goal of addressing all requirements described in the ProjectY Vision & Scope Document. Current document describes the major modules that will make up the solution within the initial scope as well as interaction and integration aspects.

This document is intended to:

- Overview high-level project description.
- Recommend technology stack.
- Give a visual representation of the relationships between application components.
- Describe delivery and operations processes.
- Describe the composition of the team and roles.

Any changes to the document after agreeing on the initial version should be tracked in the Revision history table.

1.2 Business Context

The project is an electronic medical record for urgent care and primary care, which includes clinical, billing and analytics capabilities. The product will be used in non-hospital-based urgent care clinics.

Urgent care clinics face problems such as:

- The long amount of time to input patient info;
- Increased waiting times for patients;
- The inability for clinicians to access relevant patient history;
- Inability to manage chronic medical conditions appropriately;
- Not being able to see patient recorded information at home.

The goal of the project is to fix these problems by:

- Decrease door-to-door time;
- Create streamlined workflows for the diagnoses and their corresponding treatments;
- Bring patients into the fold by allowing them to speed up their visit;
- Improve patient outcomes by providing a platform that can better manage chronic medical conditions.

1.3 Glossary

The Definitions section lists the acronyms and terms used in this document which might possess lesser familiarity or double meaning to the reader.

Abbreviation	Explanation
AWS	Amazon Web Services
SaaS	Software as a service
HIPAA	Health Insurance Portability and Accountability Act
PHI	Protected Health Information
HL7	Health Level Seven

2 Executive Summary

2.1 High Level Description

The main aim of our product is to help the clients of urgent care and primary care clinics to collect all the patient's medical data in one place. This will help to provide or to receive better and faster medical aid within the ProjectY partner clinics. Moreover, such necessary features as scheduling, advanced provision of medical information, online physician appointments and treatment receiving can be done in one single application.

From the side of the clinic, the final user can get all necessary functions in one place: appointment management, client information management, test and lab results management, e-prescriptions, telehealth, after appointment procedures, necessary communication with clients, billing and claims management. And all of these will be wrapped into the most comfortable interface with the lack of the additional and disturbing functions.

2.2 Key Decision

There are key decisions that were done during the discovery phase:

- **Usage of the AWS Cloud platform**. The AWS cloud platform was chosen because of its broad set of managed services that significantly reduce infrastructure management overheads, provide scalability and reliability;
- **All further development should be based on open-source libraries/frameworks**. By using open source technologies, the project avoids vendor lock-in and reduces licensing costs. The extensive ecosystem and community support for these tools increases development efficiency and flexibility;

- **Usage of Microservices architecture for service development at the backend**. This solution provides modularity, scalability and ease of maintenance. Each service is deployed independently of each other and focuses on a single business opportunity, such as user management, appointment scheduling or billing. It also allows individual services to scale based on demand, ensuring optimal resource utilisation;
- **Multi-tenancy SaaS approach**. The system is designed to support multiple clinics with isolated data and configuration settings, allowing each clinic to operate independently while benefiting from a common infrastructure.
- **Data Privacy and Security Measures**. Strict measures, including encryption of data at rest and in transit, have been implemented to comply with HIPAA requirements and ensure patient privacy.
- **High Availability and Fault Tolerance**. The system is designed for 24/7 availability and strategies such as load balancing, automatic disaster recovery and redundant infrastructure to manage downtime and ensure uninterrupted service. This solution is in keeping with the critical nature of healthcare services, where any system failure can directly impact patient care.
- **Integration with External Systems**. The architecture includes integration points for external systems such as laboratory results, payment processing and messaging services. By adhering to standards such as HL7 for data exchange, the system ensures compatibility and seamless interoperability with third-party vendors.

2.3 Key Risk

There are key risks that were discovered during the discovery phase:

- The system has too many integration points, as a result, the stability of the system can be disrupted by an unscheduled shutdown/outage by one of the integration services;
    - Mitigation strategy: Implement regular monitoring of third-party service status and utilise fallback mechanisms where possible.
- Integrations with Clinic API (external systems of the clinic (laboratory, file room,

etc.)) can be unpredictable in terms of unavailable docs during discovery phase;
    - Mitigation strategy: Work closely with vendors to address documentation gaps.
- High system availability is essential as any downtime can severely impact operations;
    - Mitigation strategy: Implement robust monitoring, failover and redundancy to ensure uptime.
- Protecting sensitive patient information during storage and transmission is critical to maintaining security;
    - Mitigation Strategy: Use encryption for data at rest and in transit and conduct regular security audits.
- The system must comply with strict regulations such as HIPAA which add complexity to the project.
    - Mitigation strategy: Regularly review compliance requirements and involve legal experts during design and implementation.

3 Architectural Drivers

3.1 Business Goals

ID	Business Goal Description
BG-1	Optimisation of patient information handling and clinic operations.
BG-2	Improve service speed and minimize patient waiting times.
BG-3	Provide clinicians with instant access to critical patient records.
BG-4	Foster patient satisfaction and loyalty through improved services.

3.2 Major Features

ID	Feature Description
MF-1	User-friendly interface of interaction between customers and the clinic.
MF-2	Customer data management.
MF-3	Appointment management.
MF-4	Customer notifications.
MF-5	Billing to the customer.
MF-6	Possibility of online support and acceptance of complaints.

3.3 Design Constraints

ID	Description	Priority
CT-1	The application should be in compliance with HIPAA	High
CT-2	Continuous availability must be ensured.	High
CT-3	The system must use the HL7 protocol when transmitting PHI externally.	High
CT-4	System logs and database backups must be retained for at least 5 years.	High

3.4 Architectural Concerns

ID	Description
CN-1	Logging and monitoring
CN-2	Keep customer data secured
CN-3	High level of system's availability
CN-4	HIPAA compliance
CN-5	Transferring PHI data over a network
CN-6	Transferring old patient data.

3.5 Quality Attributes and Quality Attribute Scenarios

A Quality Attribute Scenario is an unambiguous and testable requirement for one or more Solution Quality Attributes such as Performance, Usability, Maintainability and others. The scenario consists of six parts: Source of Stimulus, Stimulus, Environment, Artifact, Response, testable and accurate Response Measure.

This section lists and prioritizes the scenarios pertinent to the designed solution

ID	Quality attribute	Scenario	Business Priority	Complexity
QA-1	Security	At all times the credentials entered by the user during log-in are transferred to the server over an encrypted, secure channel without the chance of being sniffed by a third party.	High	High
QA-2	Availability	The system should be available 24/7 with 99,9% System Availability requirement.	High	High
QA-3	Permofance	The response time should be less than 1 sec	Medium	Medium

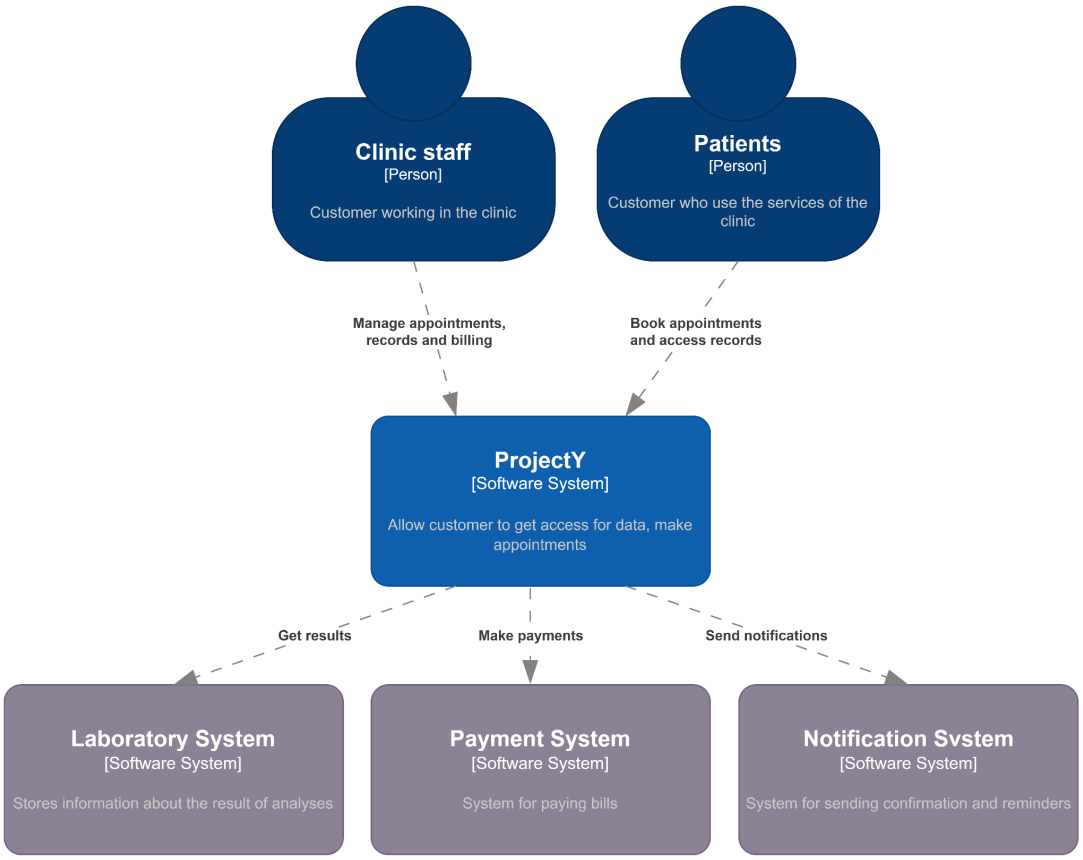
4 Solution Architecture

The section Solution Architecture is primary for the Architecture Vision document. It defines and reasons about the solution architecture design based on the architecturally significant requirements and constraints identified in the section Architectural Drivers.

4.1 System Context View

View Description: this view describes the major components of the ProjectY system as a whole.

Primary Presentation:



Element Catalog:

- Clinic Staff [Person]: Users who manage clinic operations, including appointments, billing and patient records.
- Patients [Person]: Customers of the clinic who book appointments and access their medical records.
- ProjectY [Software System]: Central application enabling clinic staff and patients to manage appointments, access records and handle billing processes.
- Laboratory System [Software System]: Stores laboratory results and provides them to the clinic.
- Notification System [Software System]: External system for sending confirmations and reminders to patients and staff.
- Payment System [Software System]: Facilitates financial transactions for clinic services.

Rationale

The Context View diagram provides a high-level representation of the system's interactions with external entities, focusing on its boundaries, responsibilities and key integrations. ProjectY serves as a centralized solution for clinic operations and patient interactions, streamlining data access and workflow for clinic staff and patients alike.

Clinic staff use the system to manage appointments, billing and patient records, ensuring operational efficiency and centralised data handling. Patients interact with the system to book appointments and access medical records, increasing engagement and reducing administrative tasks.

External integrations extend the functionality of the system. Messaging services enable timely notifications and reminders, improving communication. Laboratory systems

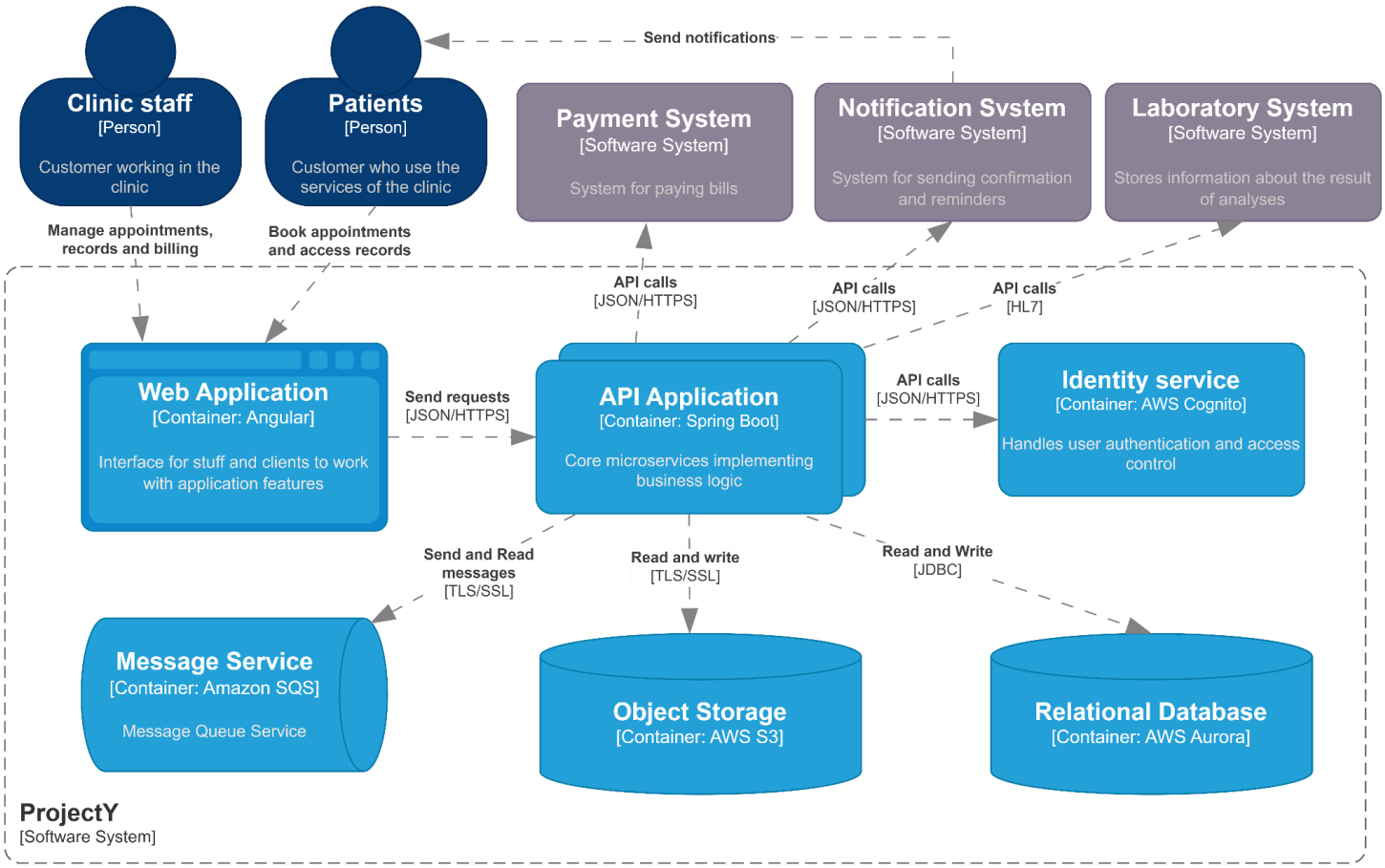
provide seamless access to diagnostic results, minimising delays. The payment system ensures secure and compliant transactions, simplifying the billing process.

The architecture ensures extensibility and interoperability, enabling smooth interaction with external services and flexibility for future enhancements. By positioning ProjectY as a central hub, the system improves communication, reduces delays and improves overall service quality.

4.2 Container View

View Description: this view describes the major modules of the ProjectY system as a whole.

Primary Presentation:



Element Catalog:

- Web Application [Container]:
  - Technology: TypeScript, Angular
  - Purpose: Provides the user interface for clinic staff and patients to manage appointments, patient records, billing and administrative tasks.
  - Interaction: Sends requests to the API Application for further processing.
- API Application [Container]:
  - Technology: Java, Spring (Boot, Cloud, Security, MVC, Data, Liquibase).
  - Purpose: Contains the core business logic and microservices to handle specific functionalities.
  - Interaction: Processes requests routed from the Web Application and interacts with the database and external systems.
- Identity service [Container]:
  - Technology: AWS Cognito
  - Purpose: Manages user authentication and authorization, ensuring secure access to application resources. Handles user registration, login, password management, and token generation.
  - Interaction: Processes requests routed from the API Application.
- Relational Database [Container]:
  - Technology: AWS Aurora
  - Purpose: Stores and manages all necessary application data, including patient records, appointments information.
  - Interaction: Read and write operations are performed by the API Application.
- Object Storage [Container]:
  - Technology: AWS S3
  - Purpose: Provides scalable object storage for storing and retrieving unstructured data, such as files, images, backups. Ensures durability, availability and security for

application assets.
    - Interaction: Read and write operations are performed by the API Application.
- Message Service [Container]:
    - Technology: AWS SQS
    - Purpose: Provides reliable and scalable message queuing for decoupling and coordinating distributed software systems and microservices. Enables asynchronous communication.
    - Interaction: Messages are sent to and received from the queue by the API Application.

Rationale

This container diagram shows the components and interactions of the system, emphasising scalability, maintainability and security to meet business objectives. The architecture balances rapid development with long-term stability.

The web application, built with Angular, provides a seamless interface for clinic staff and patients to manage operations and services. Angular ensures a responsive user experience with its modular structure and efficient data binding.

The core business logic resides in the API application developed with Spring Boot. Its built-in capabilities enable efficient RESTful API and microservices development, ensuring scalability and maintainability through modular services focused on specific functions such as appointments and notifications. Also AWS SQS allows microservices to communicate in asynchronous ways.

Data storage is managed with AWS Aurora for high performance and scalability, supported by Liquibase for easy database schema changes. This setup ensures reliability and efficiency for handling critical data such as patient records and billing. Also there is object storage for cost-optimized storing files and other unstructured data.

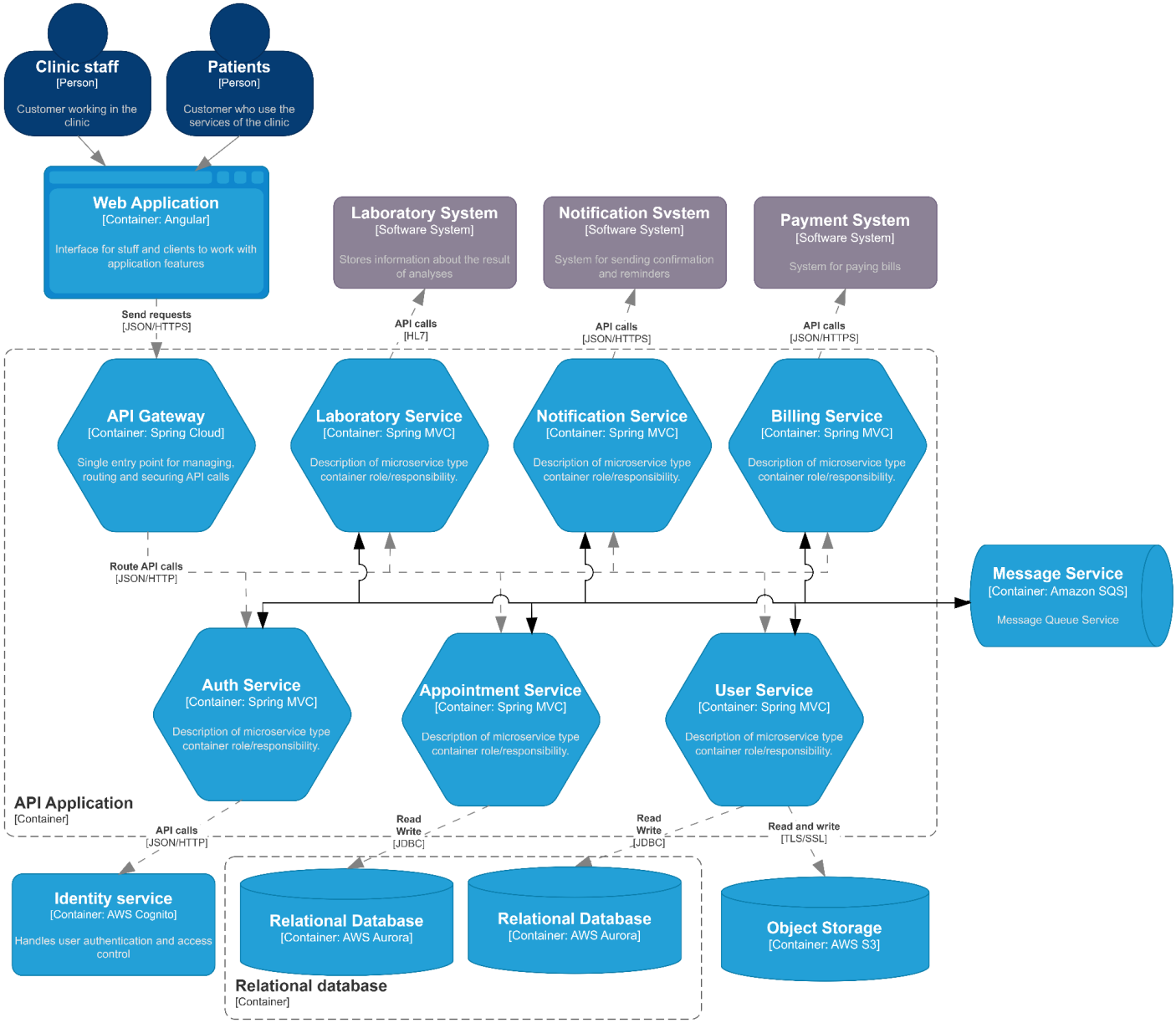
External integrations simplify the system: messaging services for notifications, lab systems for access to lab results and payment systems for secure transactions. These integrations reduce the complexity of the core application while increasing functionality.

Technologies such as the Spring ecosystem, AWS services and Angular align with project goals and ensure a scalable, secure and maintainable system. Cloud services provide high availability and cost efficiency.

4.3 Component View

View Description: The Component diagram shows how a container is made up of a number of "components", what each of those components are, their responsibilities and the technology/implementation details.

Primary Presentation:



Element Catalog:

- API Application:
  - API Gateway: Routes incoming requests to appropriate microservices.
  - Auth Service: Manages user registration, authentication and authorization.
  - Appointment Service: Handles appointment bookings and scheduling.
  - User Service: Manages user profile and settings.
  - Laboratory Service: Manages lab test records and integrates with external lab systems.
  - Notification Service: Integration service for sending email/SMS notifications via external messaging systems.
  - Billing Service: Integration service for processing invoices, payments and billing details.
- Relational databases:
  - Every microservice that requires a database, has its own dedicated database to ensure data segregation.

Rationale:

The microservices architecture is designed to ensure modularity, scalability and maintainability. The API gateway acts as a single entry point, routing requests to the appropriate microservices and handling authentication and security. Each microservice is responsible for a specific business function (e.g. user management, appointment, billing) and owns its own database, ensuring data isolation and independent scalability. This architecture improves fault isolation, scalability and service independence, ensuring a robust and adaptable system.

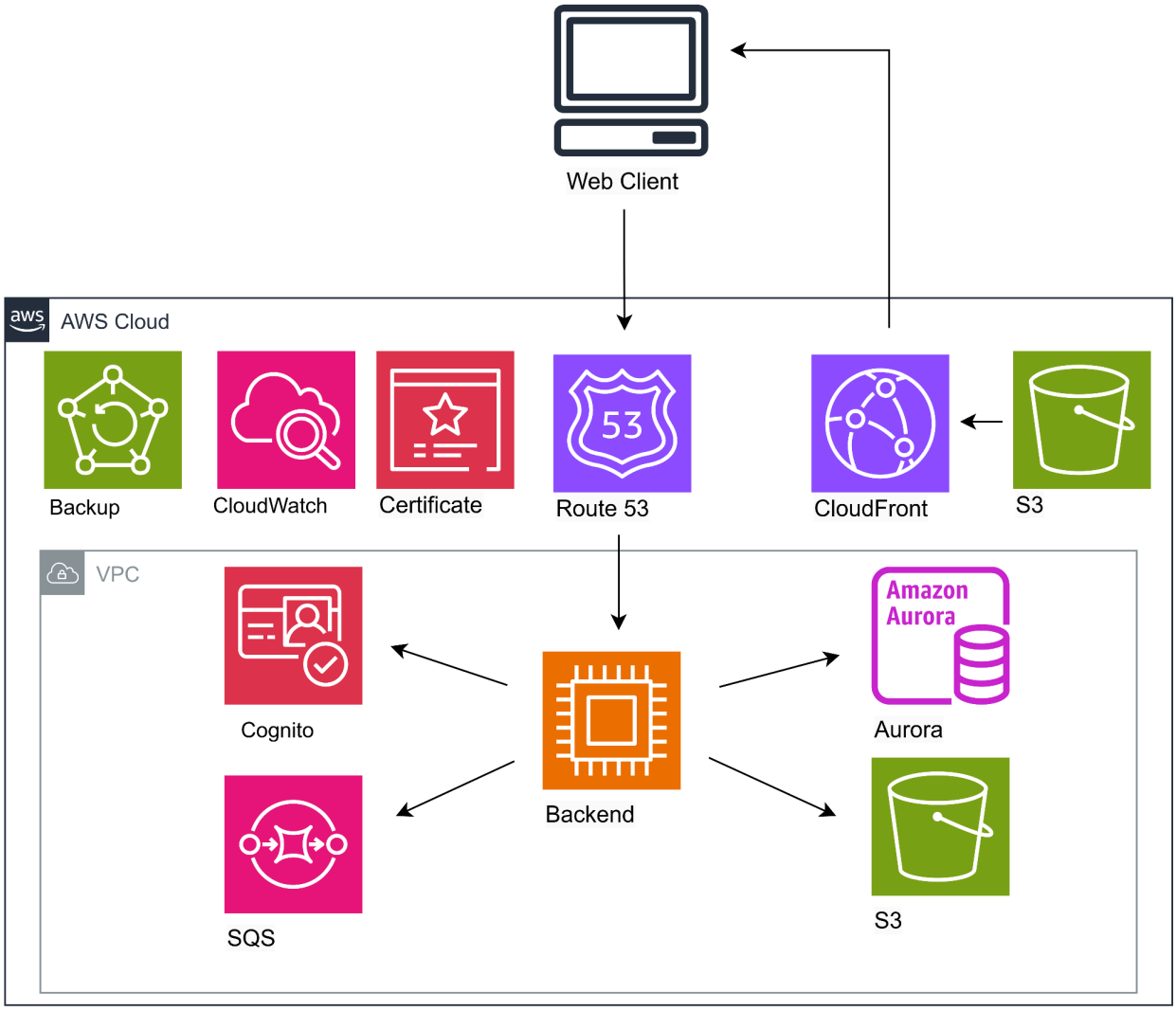
5 Operation plan

The section Solution Architecture is primary for the Architecture Vision document. It defines and reasons about the solution architecture design based on the architecturally significant requirements and constraints identified in the section Architectural Drivers.

5.1 Infrastructure

5.1.1 Infrastructure concepts

The project will be deployed on the AWS cloud platform due to its scalability, reliability and wide range of services. AWS services such as Elastic Compute Cloud (EC2), Simple Storage Service (S3) will form the foundation of the infrastructure. Additionally, AWS Aurora will be used for database management, while AWS CloudWatch will handle monitoring and logging. AWS Route 53 will ensure efficient DNS management and AWS Cognito will secure user authentication and authorization.



5.1.2 Environments

5.1.2.1 Development environment

The development environment will be designed to support continuous integration and deployment practices. It will include resources such as:

- Lightweight EC2 instances for testing and debugging code.

- AWS CodeCommit, CodeBuild and CodeDeploy for source control and automated build pipelines.
- A sandbox version of Aurora for testing database interactions.
- AWS CloudWatch for capturing logs and identifying errors.

5.1.2.2 Staging environment

The staging environment will replicate the production setup as closely as possible to facilitate pre-release testing. Features include:

- Identical EC2 instance configurations to production.
- A duplicate Aurora database to test data integrity.
- Integration with AWS CloudWatch to monitor system behavior under load.
- Route 53 for testing DNS routing.

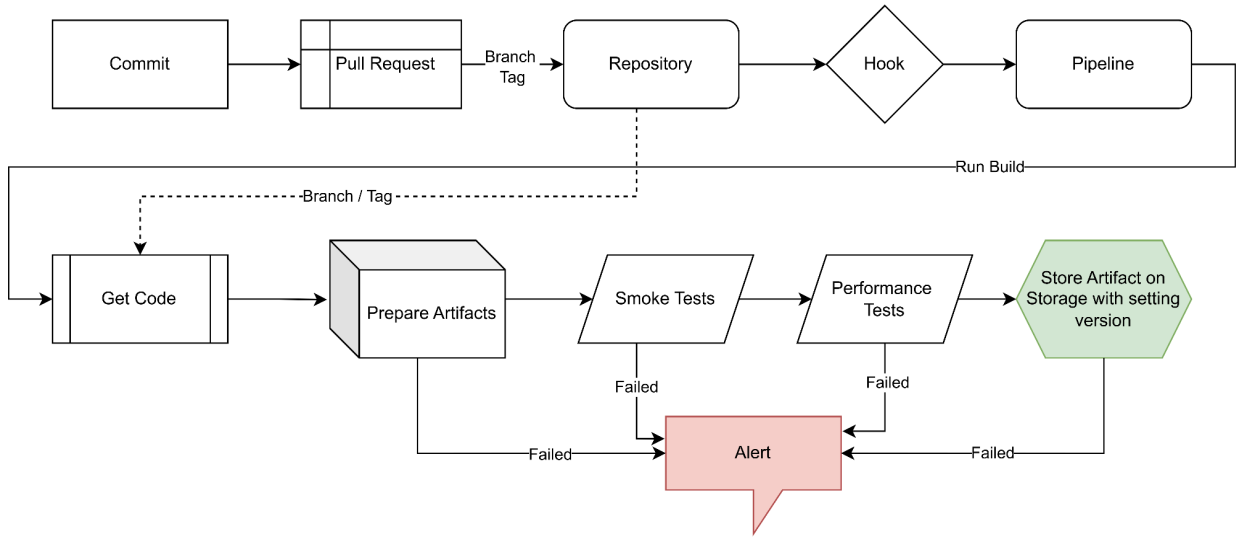
5.1.2.3 Production environment

The production environment will provide a robust and scalable setup to handle real-world traffic and operational requirements. Key components include:

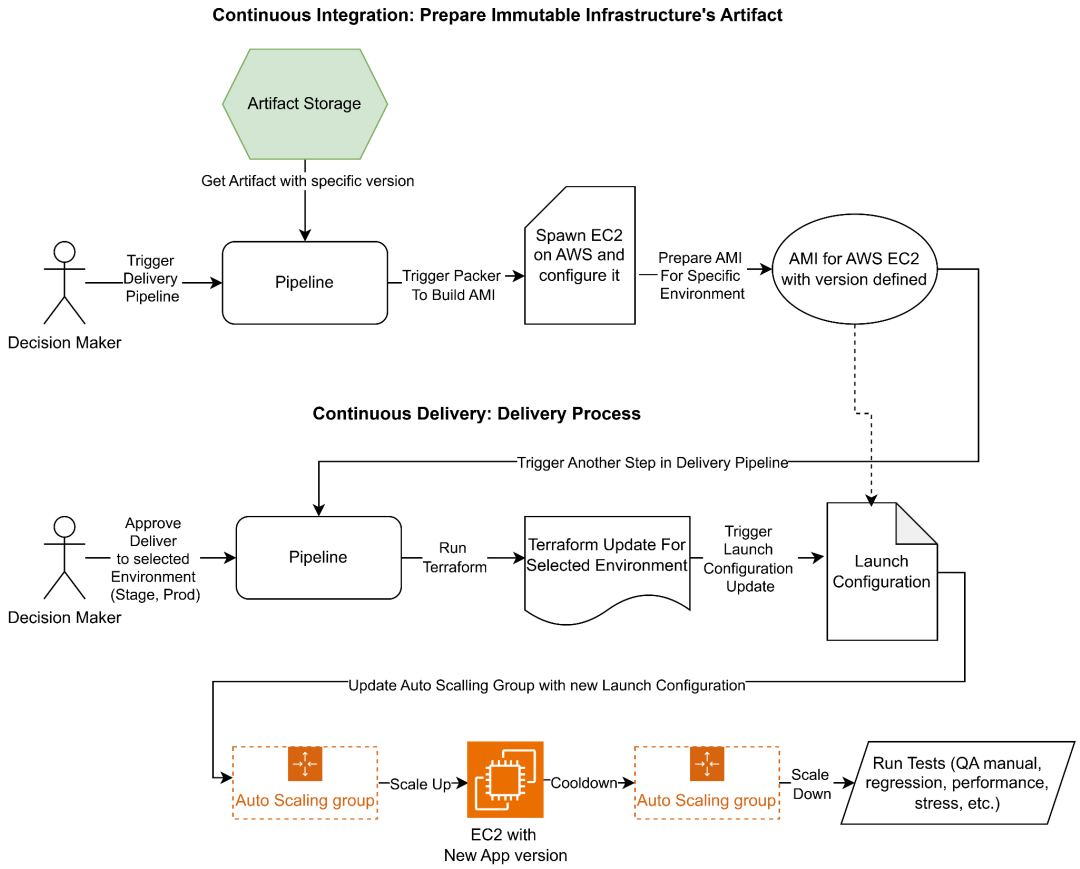
- High-performance EC2 instances to support application workloads.
- AWS Aurora for secure, reliable database management.
- AWS CloudWatch for proactive monitoring and alerting.
- Route 53 for DNS management.
- AWS Cognito for user management and security. This environment will be monitored and maintained to ensure high availability, minimal downtime and quick recovery from failures.

5.2 Continuous integration and continuous delivery

A CI/CD pipeline is an automated process that integrates, tests, and deploys code changes to production environments. It ensures consistency, reduces errors, accelerates software release, and improves quality. By providing immediate feedback and fostering collaboration, it helps deliver reliable, high-quality software quickly and efficiently.



The CI process starts with a commit, followed by a pull request tagged with a branch. This branch is then pushed to a repository. A hook triggers the pipeline to run the build, involving steps like getting the code, preparing artifacts, and running smoke and performance tests. If any tests fail, an alert is generated. If all tests pass, the artifact is stored in storage with a set version.



The process begins with a decision maker initiating the pipeline, which retrieves an artifact version and triggers Packer to build an Amazon Machine Image (AMI) using a configured AWS EC2 instance. Upon creation, the AMI is approved for delivery to the desired environment, where Terraform updates the environment, modifies the launch configuration and auto-scaling group and scales EC2 instances with the new app version. The process concludes with various tests to ensure quality and performance, thus maintaining a structured, automated approach for consistent and reliable deployment.

5.3 Monitoring

5.3.1 Log collection

Application and system logs will be collected using a centralized logging solution, enabling efficient storage, querying and analysis.

5.3.2 Application monitoring

Workload metrics such as request rates, response times and error rates will be tracked to ensure performance and reliability.

5.3.3 Environments and infrastructure monitorings

Infrastructure health, including CPU, RAM, disk usage and network performance, will be continuously monitored to identify and resolve issues promptly.

5.3.4 Notifications

Alerts and notifications will be configured for critical events using email, SMS or integrated DevOps tools.

6 Implementation Roadmap

6.1 Implementation Milestones

ID	Business Goal Description	Success Criteria Description
M-0	Optimisation of patient information handling and clinic operations.	Patient data is centralized and accessible across clinics, eliminating redundant input and improving operational efficiency.
M-1	Improve service speed and minimize patient waiting times.	Patients can book appointments and receive services faster, with average waiting times reduced by at least 30%.
M-2	Provide clinicians with instant access to critical patient records.	Clinicians can retrieve and review complete patient histories within seconds through a secure and user-friendly interface.
M-3	Foster patient satisfaction and loyalty through improved services.	Positive patient feedback increases by 20% Patient retention rates improve due to seamless and efficient interactions.

6.2 Technology Stack

Technology	Function
Backend	
Java, v. 21 (LTS) (ex. OpenJDK v.21.0.x)	Programming platform and web engine
Spring Boot	A framework for building production-ready applications quickly and easily with Java
Spring Security	Provides comprehensive security services for Java applications, including authentication and authorization
Spring Data JPA	Simplifies data access using the Java Persistence API, reducing boilerplate code
Spring Actuator	Adds production-ready features to your application, such as monitoring and managing health and metrics
Spring Cloud	Tools for building common patterns in distributed systems, like configuration management and service discovery
jUnit	A framework for writing and running unit tests in Java

Mockito	A mocking framework for unit tests in Java, allowing you to create and test mock objects
Liquibase	A database schema change management solution that helps track, version and deploy database changes
Frontend	
TypeScript	A static type checker, used by Angular
Angular	A platform and framework for building single-page client applications using HTML and TypeScript
Jasmine	For unit testing in Angular
DevOps	
AWS Aurora	A fully managed relational database service providing high availability, scalability and compatibility with PostgreSQL and MySQL.
AWS Cognito	Provides user sign-up, sign-in and access control for web and mobile applications.
AWS Route 53	A scalable and highly available Domain Name System (DNS) web service for routing traffic to applications.

AWS NAT Gateway	A managed service that allows private subnets in a Virtual Private Cloud (VPC) to connect to the internet securely.
AWS Backup	Fully managed backup service centralizing and automating the backup of data across AWS services.
AWS Virtual Private Cloud	Enables the creation of logically isolated virtual networks in the AWS cloud, ensuring secure and scalable deployments.
AWS CodeCommit	A fully managed source control service for hosting secure Git repositories.
AWS CodeBuild	A fully managed build service that compiles source code, runs tests and produces ready-to-deploy software packages.
AWS CodeDeploy	Automates application deployments to various compute services like EC2 and on-premises servers.
AWS CodePipeline	A continuous integration and delivery (CI/CD) service for automating release pipelines.
AWS CloudWatch	A monitoring and observability service providing data on resource utilization, application performance and operational health.

6.3 Team Skillset

Role	Skillset
Solutions Architect	• Java 21 • AWS Platform (EC2, Aurora, S3) • Microservices Architecture • Spring Framework • REST API Design • Kubernetes and Docker • System Design and Scalability • CI/CD • Security Best Practices • Monitoring Tools
Senior Backend Engineer	• Java 21 • AWS Platform • Spring Boot • Spring Cloud • REST • Git • Maven
Middle Backend Engineer	• Java 21 • Spring Boot • AWS Basics • Git • Maven
Senior DevOps Engineer	• AWS • Terraform
Senior Frontend Engineer	• TypeScript • Angular • Jasmine • Responsive and Accessible Design
Middle Manual QA Engineer	• TestRail • Postman • Jira

Business Analytic	• Confluence • Jira
Project Manager	• Confluence • Jira
UI/UX Designer	• Figma • Adobe XD

6.4 Team Structure

Role	Responsibility	Count	FTE
Solution Architect	• System analysis and design • System requirements specification • Technical leadership and communication	1	0.5
Senior Backend Engineer	• Technical communication • Code reviews • Backend Core functionality implementation • Writing tests	1	1
Middle Backend Engineer	• Backend Core functionality implementation • Writing tests	1	1
Senior Frontend Engineer	• Frontend implementation • Writing tests • Technical communication	1	1
Middle Manual QA Engineer	• Writing and executing test cases • forming test reports • Bug management	1	1
Business Analytic	• User stories creation • Documentation creation	1	0.5
Project Manager	• Team organization • Schedule management • Risk management • Progress reporting	1	0.5
UI/UX Designer	• Prototyping • User interface and experience design	1	0.5