Colombo International Book fair Stall Reservation System

Software Architecture Document

Version <1.0>

Colombo International Book fair Stall Reservation System	Version: <1.0>
Software Architecture Document	Date: 24/10/2025
RATHNAYAKE R.M.B.T.M. EG/2020/4162	

1. Background

1.1 Problem Statement

The Colombo International Book Fair has noticed rising interest for stall reservations, making the manual process inefficient and prone to errors. The system must digitize the registration, stall selection, and confirmation procedures for book publishers and vendors.

1.2 Intended Solution

A web-based Reservation Management System to allow publishers and vendors to register, browse the exhibition map, reserve up to 3 stalls per business, receive confirmation emails containing a unique QR pass, and manage genres. An employee-only portal allows organizers to view stall availability and reservation details.

Two separate frontends: (1) Public Publisher/Vendor Portal and (2) Employee Portal, both built in React. The backend is implemented using Spring Boot (Java) exposing REST APIs. MySQL serves as the relational database. Supporting services include email delivery, QR generation, and file/object storage for QR images.

1.3 Main Stakeholders and Concerns

Table 1: *Stakeholders* and their Concerns

Main Stakeholder	**Primary Concern(s)**	**Architectural Implication**
Book Publisher/Vendor	Usability (easy reservation process, clear map) Reliability (receiving confirmation/QR code) Security (data privacy)	Requires a highly available, fast Vendor Portal and reliable Asynchronous Notification worker.
Organizer Employee	Accuracy (real-time stall status) Security (restricted portal access) Control (ability to manage/view all reservations)	Requires secure Employee Portal and strict Authentication and Authorization at the Backend API level.

2 Requirements

2.1 Functional Features

I. User registration and login.
II. Stall browsing via an interactive exhibition map.
III. Reservation of up to 3 stalls per business.
IV. Confirmation email with unique QR code as a digital entry pass.
V. Genre management on the home screen.
VI. Employee portal for organizers to view and manage reservations.
VII. REST API backend and separate frontends.

2.2 Quality Attributes and Scenarios

Table 2: Quality Attributes Scenarios

Quality Attribute	**Refinement**	**Scenario**	**Response/Measure**
Availability	Fault Tolerance	Source: Vendor portal web server fails. Stimulus: A vendor attempts to register.	Response: A load balancer redirects the request to an available server instance. Measure: Less than 5 seconds of downtime for the Vendor Portal during peak hours.
Performance	Latency (Map Load)	Source: Vendor accessing the reservation page. Stimulus: Vendor requests the stall map view.	Response: The system displays the interactive map. Measure: Map loads and updates within 3 seconds.
Security	Authentication	Source: An unauthorized user (not an organizer).	Response: The Authentication Service denies access with a HTTP 401 response.

		Stimulus: Attempts to access the Employee Portal.	Measure: 100% of unauthorized attempts must be rejected at the API Gateway/Auth Service.
Scalability	Concurrency	Source: Multiple vendors. Stimulus: 50 concurrent requests to reserve a single available stall.	Response: Only the first successful request is processed; others receive a "Stall Unavailable" message. Measure: The system handles 100 requests per second without transaction errors.

2.3 Assumptions

- Venue map and stall data are provided as JSON/SVG overlays.
- No online payments in the current version.
- JWT-based authentication for users and employees (RBAC roles).
- Peak traffic expected at opening of reservation period; scaling required.
- Tech stack must align with Java/Spring Boot expertise for backend.

3 Architectural Design

3.1 Architectural Design Patterns

The architecture follows a Layered Architecture pattern and a Client-Server pattern with a specific focus on stateless backend services to ensure horizontal scalability.

Table 3: Architectural Design Patterns

Design Pattern Name	**Problem(s) Satisfied**	**Tradeoffs**
Layered Architecture	Separation of concerns, maintainability, testability.	Can introduce performance overhead (latency) due to multiple layers of communication.
Client-Server (REST API)	Decoupling of front-end and back-end, allowing multiple clients (Vendor/Employee).	Requires robust API documentation and version control.

3.2 Stateless Scalability

- The Backend API and all application servers will be stateless**.** User session data will be maintained via JSON Web Tokens (JWT)**.**

  This is the key approach to achieving Scalability and Availability**.** During high reservation traffic, stateless servers are easily load-balanced and horizontally scalable using services like AWS Auto Scaling Groups/ECS**.** This ensures no session data is lost if an instance fails.

3.3 Event-Driven Confirmation (Asynchronous Processing)

- To handle post-reservation tasks **(**QR creation and email sending**),** use an asynchronous technique via a Worker Pool and AWS SQS Message Queue.

This increases Performance (Latency)**.** The vendor receives immediate booking confirmation from the API. The heavy, I/O-bound operations (like connecting to AWS SES and QR image generation) are deferred to the background QR/Email Worker, preventing the main Spring Boot API thread from becoming a bottleneck and maintaining a fast user experience during high concurrency.

3.4 C4 Model Views

3.4.1 C1: Context Diagram

The Context diagram shows the system as a single entity and how it interacts with the outside world (users and external systems).

The diagram shows the central Stall Reservation Management System box in the middle. It connects to two people: Book Publisher/Vendor and Organizer Employee, and one external system: Email/QR Service**.**

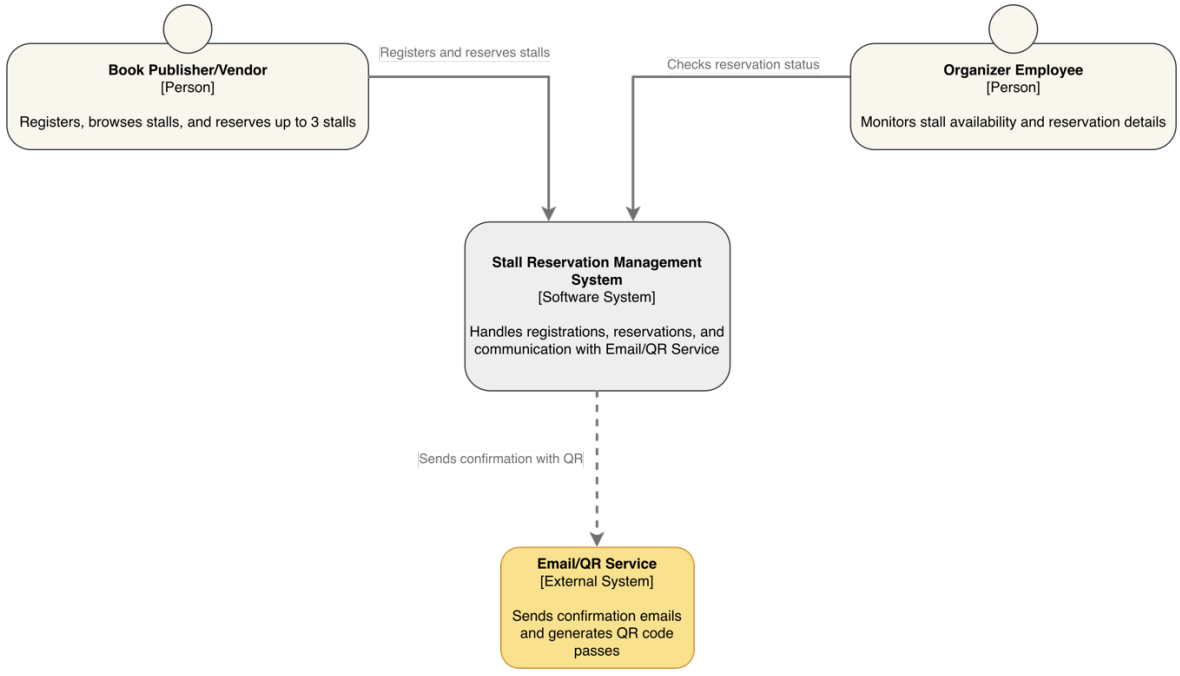

Figure 1: C4 Context Diagram

3.4.2 C2: Container Diagram

The Container diagram decomposes the system into deployable applications (frontend, backend) and data stores.

The system is decomposed into four main containers: the Bookfair User Portal and Organizer Employee Portal, which communicate with the central Reservation API (Backend). The API interacts with the Reservation Database. Finally, the API communicates with the external Email/QR Service**.**

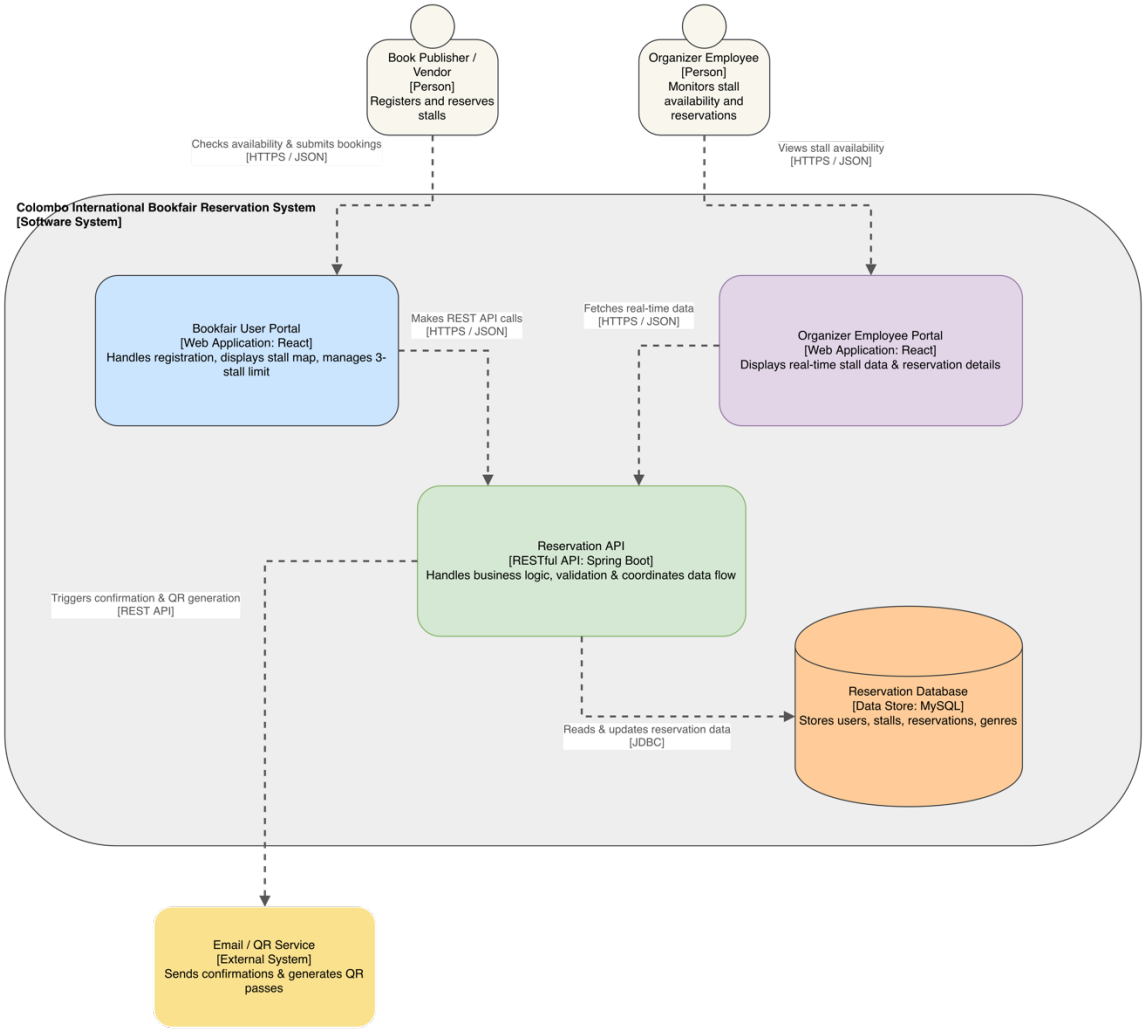

Figure 2: C4 Container Diagram

Table 4: Technologies

Element Name	Type	Technology	Brief Description
Vendor Portal	Web Application	React	Provides user interface for vendor registration and stall reservation.
Employee Portal	Web Application	React	Provides user interface for employee/organizer stall availability checks.
Backend REST API	Application	Spring Boot (Java)	Stateless, central application containing all business logic and orchestrating data persistence.
Reservation Database	Data Store	MySQL (AWS RDS)	Relational database for transactional data (stalls, users, reservations, genres).
Notification Queue	Infrastructure	AWS SQS	Decouples the API from the worker, ensuring reliable, asynchronous processing of confirmations.
QR/Email Worker	Worker	Spring Boot (Java) + ZXing Library	Consumes messages from AWS SQS. Uses ZXing for QR generation and sends email via AWS SES.
External Email System	External System	AWS SES	External AWS service responsible for high-volume transactional confirmation emails.

3.4.3 C3: Component Diagram

The C4 Component Diagram explains how the Bookfair Reservation System is divided into smaller parts that work together. The frontend includes the Bookfair User Portal for vendors to register, log in, and reserve stalls, and the Organizer Employee Portal for organizers to view stall details and manage reservations. Both portals communicate with the backend through secure RESTful APIs.

The backend is managed by the Reservation API, which handles business logic, data validation, and coordination with the Reservation Database (MySQL). It also connects to an Email and QR Service to send booking confirmations and generate digital passes. This structure keeps the system modular, reliable, and easy to maintain.

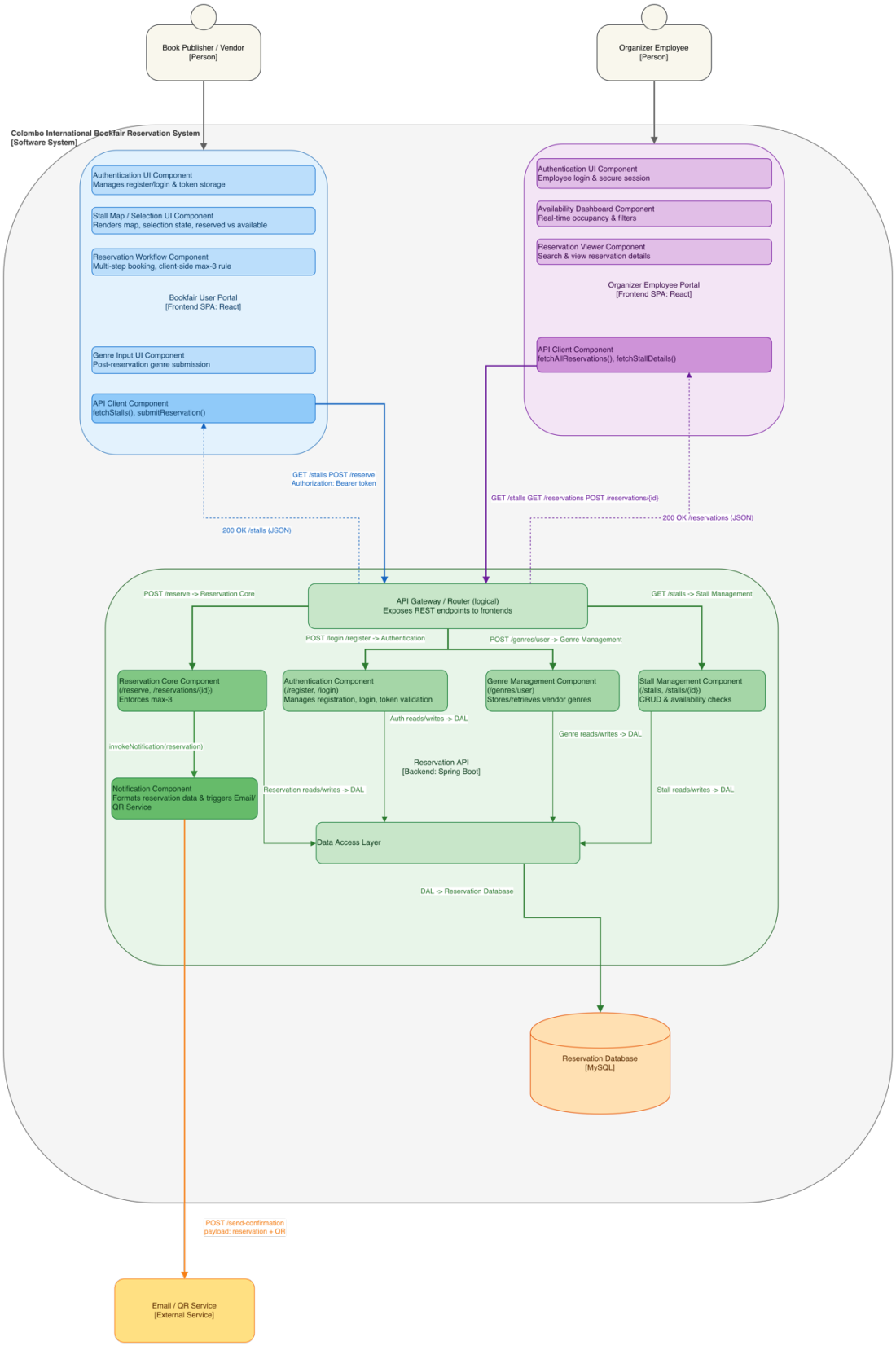

Figure 3: C4 Component Diagram

Zooms in on an important container (the Backend API) to show its internal structure, service breakdown, and component roles

Table 5: Backend components

Component	**Responsibility**	**Interface/Protocols**	**Collaborators**
Authentication Component	Manages user registration, login, and token validation.	REST Endpoints (/register, /login)	Reservation Database
Stall Management Component	CRUD operations for stall data (size, dimensions, name) and real-time availability checks.	REST Endpoints (/stalls, /stalls/{id})	Reservation Database
Reservation Core Component	Executes the booking process. Enforces the business rule (max 3 reservations per user). Handles transactionality of booking (check, lock, save).	REST Endpoints (/reserve, /reservations/{id})	Reservation Database, Notification Component
Genre Management Component	Stores and retrieves the literary genres a vendor intends to display (post-reservation step).	REST Endpoints (/genres/user)	Reservation Database
Notification Component	Formats reservation data (including QR code payload) and calls the external email service.	External RPC/HTTP	Email/QR Service

Component Flow (Reservation)

1. The Bookfair User Portal sends a POST /reserve request to the Reservation Core Component.
2. Reservation Core Component queries the Stall Management Component and Reservation Database to check current reservations for the user.
3. If the rules are met, the Reservation Core Component creates the new reservation record in the database.
4. It then calls the Notification Component with the reservation details.
5. Notification Component communicates with the external Email/QR Service to send the final confirmation email.

3.4.4 C4: Code Diagram (Focus: Reservation Component)

Zooms in on an important component (the Reservation component) to show its internal structure

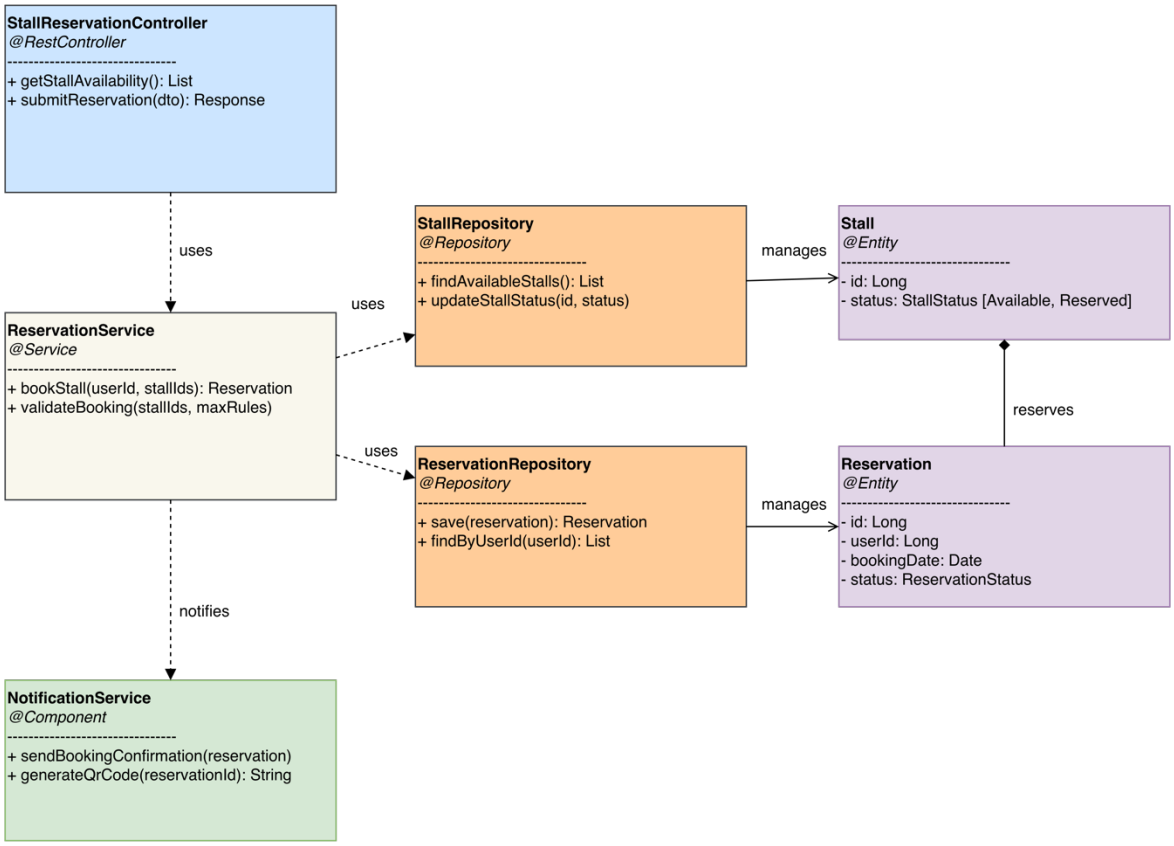

Figure 4: C4 Code Diagram(Reservation Component)

This UML Class Diagram visualizes the core domain entities (Stall, Reservation) and the primary service class (ReservationService) within the backend, including its necessary dependencies (ReservationRepository for data access and the Notifier interface for external communication).

3.5 Deployment View

This view describes how the software containers are mapped onto the production infrastructure to ensure Availability and Scalability. Assume a standard AWS Cloud Environment with Jenkins for CI/CD.

3.5.1 Conceptual Deployment Structure (AWS & CI/CD)

Table 6: Element Catalog for Deployment

Element Name	Type	Brief Description
Jenkins Server	CI/CD Server	Automates the build, test, and deployment process onto the AWS infrastructure.
Load Balancer (AWS ALB)	Infrastructure	Distributes traffic across multiple application instances for availability and performance.
Application Cluster (AWS ECS/EKS)	Infrastructure	Container orchestration running multiple replicas of the Frontend and Backend applications.
Database Cluster (AWS RDS)	Infrastructure	Primary MySQL instance with one or more replicas for high availability and read-scaling.
Notification Queue Cluster (AWS SQS)	Infrastructure	AWS Simple Queue Service used as the high-availability message broker.
Worker Pool (AWS ECS)	Infrastructure	Set of auto-scaling resources dedicated to the asynchronous QR/Email Worker.

This view describes how the software containers are mapped onto the production infrastructure to ensure Availability and Scalability.

4 References

[1] *Google Drive Document*, [Online]. Available: https://drive.google.com/file/d/1eT9RdHSdaYyt_D2RpmhjzhcfKGjfPidF/view . [Accessed: 24–Oct–2025].

[2] S. Brown, “Introduction | C4 model,” *c4model.com*, 2018. [Online]. Available: https://c4model.com/. Accessed: Oct. 24, 2025.