# System design overview

*Cloud Sales System – Crayon*

# Table of contents

[1. Introduction 2](#_Toc195457463)

[1.1 Purpose 2](#_Toc195457464)

[1.2 Scope 2](#_Toc195457465)

[1.2 Definitions, Acronyms and Abbreviations 2](#_Toc195457466)

[1.3 Context 2](#_Toc195457467)

[2. System design 3](#_Toc195457468)

[2.1 High Level System Overview 3](#_Toc195457469)

[2.2 Data consistency between Regions 4](#_Toc195457470)

[RW database with cold-start fail-over database 5](#_Toc195457471)

[Dedicated primary RW database, with secondary RO 6](#_Toc195457472)

[2.3 Application Components 7](#_Toc195457473)

[Product management API 8](#_Toc195457474)

[Customer Management API 9](#_Toc195457475)

[Account/Billing API 9](#_Toc195457476)

[Licence/Subscription API 9](#_Toc195457477)

[Order API 9](#_Toc195457478)

[Configuration API 10](#_Toc195457479)

[Notification Dispatcher 10](#_Toc195457480)

[Invoice Processing Worker 10](#_Toc195457481)

[Order Processor 10](#_Toc195457482)

[Document Generator 11](#_Toc195457483)

[Email Dispatcher 11](#_Toc195457484)

[Use cases 12](#_Toc195457485)

## Introduction

**Project name**: Cloud Sales System

### 1.1 Purpose

This System Design Document (SDD) has been prepared as part of an assignment to demonstrate the system overview of **Cloud Sales System**.

### 1.2 Scope

The intended audience: Crayon

### 1.2 Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Definition |
| CSS | Cloud Sales System |
| CCP | Cloud Computing Provider |
| API | Refers to a set of component/system Endpoints |
| HTTP | Hypertext Transfer Protocol |
| MS | Microservice |
| Gateway/BFF | Main entry point of external systems (web apps, other systems) |
| SLA | Service-Level Agreement |
| AKS | Azure Kubernetes Service |
| AZ | Availability Zone |
| RW | Read-Write |
| RO | Read-Only |

### 1.3 Context

Crayon wants to implement a solution for cloud sales, which will serve customers in Europe and APAC. Crayon has a business partner, a Cloud Computing Provider (CCP). The CCP offers an API that Crayon can use to automate the business.

Crayon wants to create a way to sell these services to its customers in two ways:

1. A web portal where Crayons customers can log in and perform the actions.

2. A web API where Crayons customers can integrate their systems, and perform the same operations without any user interaction. System to System.

Crayon wants to provide exactly the same set of services in both of the above cases. A customers should be able to choose to use alt 1 or alt 2, or use a combination of 1 and 2.

Primary challenges:

* Entry point for different types of clients (Web applications and services)
* Provide exactly the same set of services for any client
* Data consistency in multi-region deployment

## System design

### 2.1 High Level System Overview

Key challenges considered while designing this system:

* Data consistency and replication
  + Ensuring that all data is available and up-to-date in every region. This is done by continuous geo-replication.
  + Having Primary (Read-Write) database and Secondary (Read-only) is requirement multi-regional systems. When Primary region fails (or primary database fails), Secondary database is promoted to Primary, and all write requests are re-routed to the new Primary.
* Service latency and performance
  + Multi-regional services need to ensure low latency for users from different geographical locations. The goal is to route traffic to the closest region (or the least-loaded) to minimize delays. Azure Front Door is one of the possible tools for geo-aware traffic routing, load balancing.
* Availability and fault tolerance
  + To ensure high availability, proposed solution is to deploy application services into multiple AZs
  + Azure Storage blob is replicating storage into three AZs by default
* Disaster recovery/Backup
* Security and access control
  + Each service must use modern authentication/authorization mechanisms. Proposed solution is Microsoft Entra ID for Identity and access management, and it’s part Microsoft Entra External ID (B2C) for management of end clients (customers)
* Monitoring and logging
  + Azure monitor can aggregate data from multiple regions

Solution of system design overview can be seen in the following diagram.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1 High Level Overview - Cloud Sales System

### 2.2 Data consistency between Regions

When building a multi-regional application, having data replicas in each region is important for two key reasons:

* Low Latency for Users Worldwide
  + Replicas ensure that users in **Europe**, **APAC** can access data quickly from the closest region.
  + This reduces network round-trip time and improves app responsiveness
* High Availability and Resilience
  + If a region goes down (due to disaster or Azure outage), another region can take over without data loss.

Two strategies can be implemented:

* Dedicated Primary RW database in one region, consumed by apps from both regions
* Dedicated Primary RW database in one region, with RO database in secondary region

For both scenarios, Geo-Replication between databases must be configured.

#### RW database with cold-start fail-over database

Following figure shows when Region A healthy.

* Database is Primary RW.
* Geo-Replication is configured to fail-over database in secondary Region

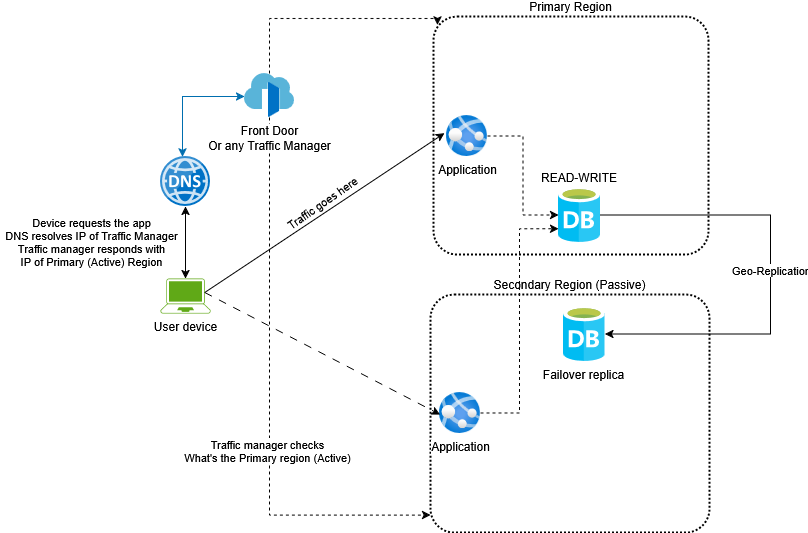


Figure 2 Healthy Primary Region

This figure shows scenario when Primary region fails:

* Secondary replica becomes Primary RW, all traffic is routed to Secondary Region (now Primary)

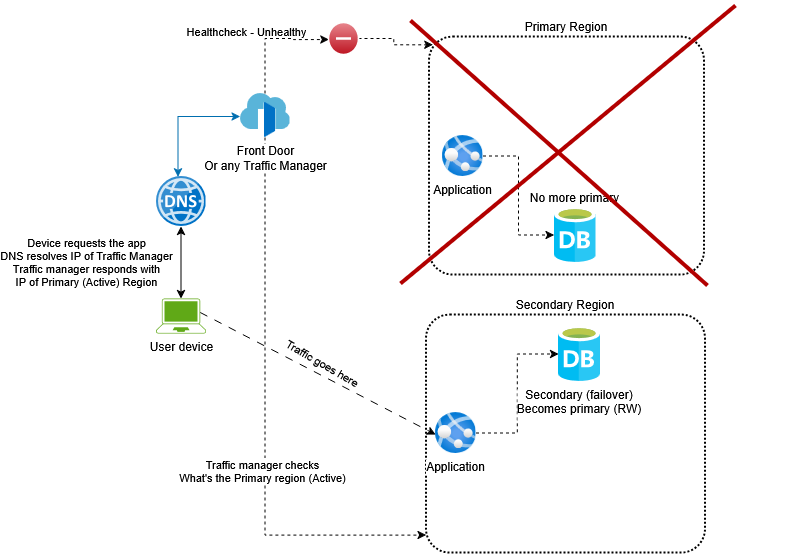


Figure 3 Primary region is not available

#### Dedicated primary RW database, with secondary RO

In this scenario, there are two active databases:

* Primary RW database
* Secondary RO database
* Geo-Replication is configured

Applications write to primary (RW) database, and read from the closest RO database.

If required, Secondary can be promoted to Primary (RW), while old RW can become secondary (RO) database.

The following Figure shows diagram for this scenario

A diagram of a computer network

AI-generated content may be incorrect.

Figure 4 RW Primary database + RO Secondary database

### 2.3 Application Components

One of the business requirements is that the application services should be consumed by Web Frontend and by external systems (services).

Proposed solution is implemented as a set of microservices with the main service (Gateway).

Gateway is a entry point of the system. It should be accessed by any type of client via HTTP/S protocol. Gateway also can be easily configured to limit backend services access and acts as a Façade.

With microservice architecture, implementation can be separated between multiple domains:

* Product management
* Customer Management,
* Account management
* Licence Management
* Document management
* Business configuration
* …..

In the following Figure you can find proposed solution components.

A diagram of a company

AI-generated content may be incorrect.

Figure 5 Component overview

#### Product management API

**Purpose**: Manages the catalog of cloud services or products being sold.

* CRUD operations for services/products/plans
* Handles pricing tiers, feature sets, bundles
* Provides catalog info for other services (e.g., Web Portal, External Systems, Order API)
* Configurable to access, locally manage products/services retrieved from any kind of web services, acts as Adapter
* Pushes event messages related to product changes.

#### Customer Management API

**Purpose**: Manages customer identity and profile data.

* Create/update customer records
* Links customer with accounts, subscriptions
* Stores contact info, preferences, legal details
* May integrate with identity providers (e.g., Azure AD B2C)

#### Account/Billing API

**Purpose**: Manages billing accounts and payment-related data.

* Creates billing accounts for customers
* Handles payment method management (cards, invoices)
* Calculates charges, stores billing history

#### Licence/Subscription API

**Purpose**: Tracks customer subscriptions and license usage.

* Assigns subscriptions to customers/accounts
* Manages license allocation, limits, renewals
* Can check for license availability or usage compliance

#### Order API

**Purpose**: Handles order placement and checkout flow.

* Accepts orders for products/services
* Validates customer/account details
* Billing, and fulfillment workflows
  + Charge the customer, sends events for invoice generation
  + Deliver the licences, sends events, confirmation email..
* Tracks order status (pending, processing, completed)

#### Configuration API

**Purpose**: Manages global or tenant-level configuration settings.

* Feature flags, pricing overrides…
* Regional branding, rules, regional preferences
* Used by frontend or other services to adjust specific behavior

#### Notification Dispatcher

**Purpose**: Converts domain events into user-facing notifications.

* Subscribes to events (e.g., InvoiceGenerated, SubscriptionActivated, SubscriptionCanceled…)
* Filters and maps them to end-user readable messages

#### Invoice Processing Worker

**Purpose**: Automates billing cycles and invoice generation.

* Listens for billing events
* Calculates usage-based or fixed charges
* Generates invoices and records them in billing system
* Triggers events: InvoiceGenerated …

#### Order Processor

**Purpose**: Orchestrates order fulfillment.

* Picks up new or pending orders from Order API (via event bus)
* Validates stock/availability (via Products API)
  + For faster communication between services, gRPC can be easily integrated
* Updates order status and raises events (e.g., OrderFulfilled)

#### Document Generator

**Purpose**: Creates downloadables (e.g., PDFs, receipts, agreements).

* Generates invoices, license agreements, welcome letters…
* Pulls data from billing, customer, and subscription APIs
* Stores documents as attachments with downloadable links inside Blob storage

#### Email Dispatcher

**Purpose**: Handles sending actual emails.

* Listens for SendEmail events
* Integrates with SMTP providers
* Manages retries, delivery tracking…

## Use cases

Sequence diagram showing example of Get Products From CPP flow

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 6 Get products from CPP flow example