iSyncRabbit – Integration Hub

by
Devrabbit IT Solutions

Table of contents

1. Introduction

1.1 Objective

The objective of this application **(ISYNC)** is to deliver a unified and scalable integration platform that enables seamless data flow between two / more independent service management systems. This application eliminates operational fragmentation by automating cross-platform data synchronization through scheduled background jobs, enhancing efficiency, accuracy, and visibility without manual intervention.

1.2 Target audience

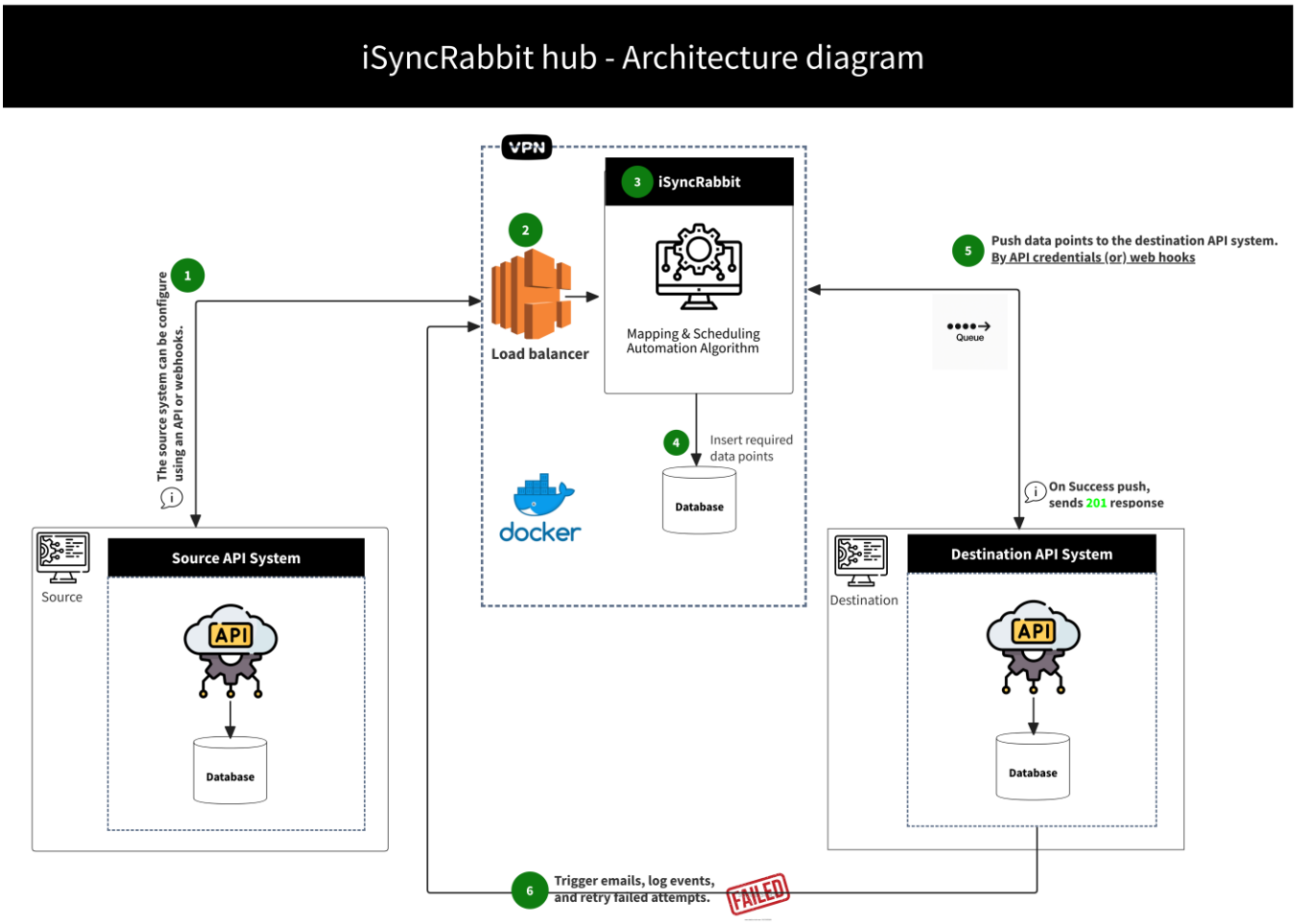
Target Audience
This application is designed for corporate real estate organizations that manage large, distributed portfolios and operate across a variety of service and operational systems—such as CMMS, IWMS, ERP, and project management platforms. It enables seamless data integration across these disparate systems, eliminating manual effort and delivering real-time visibility and consistency across the entire real estate operation.

1.3 Technologies stack

The application is built using the MERN stack, supported by additional tools for orchestration and deployment:

Front-end	React JS & Redux
Backend / API	Node JS
Database	NO-SQL database
DevOps Environment	Docker, Pipelines
Cloud system	AWS cloud
Networking hosting	AWS Load Balancer

2. Architecture Diagram



3. Application overview

3.1 Create Integration between two systems

Process of Create integration requires user to go through five important steps as following :-

- **Title and Description**

  This step creates first instance of all the five steps in the database with the option to describe Integration.

- **Source API credentials**

  This step is for the user to select the source system among the provided and enter its valid API credentials.

- **Destination API credentials**

  This step is for the user to select the destination system among the provided and enter its valid API credentials.

- **Destination to Source ‘s Field mappings**

  In this step, users define the desired field mappings for its respective method(POST, PATCH) so that data in the destination system is accurately aligned with corresponding fields from the source system's records by the Mapping & Scheduling Automation Algorithm.

- **Update settings of integration**

  This step allows user to configure its frequency, desired date range, status mappings etc;

4. Environments Overview

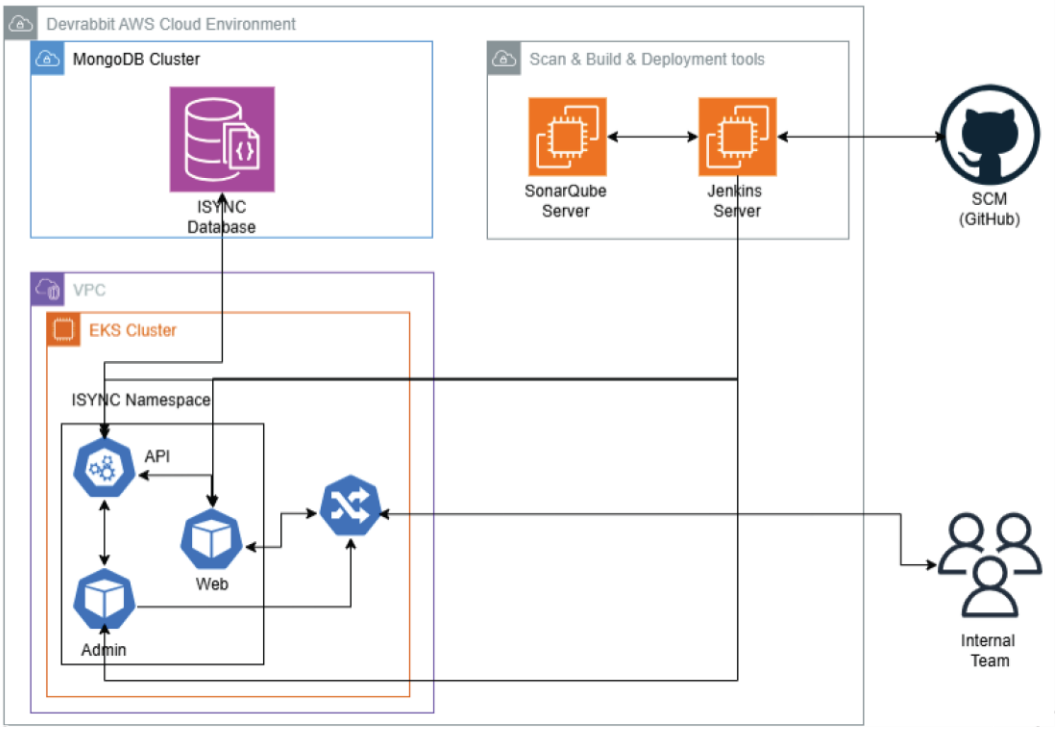
Two isolated environments are maintained:

Environment	Description	AWS Region	Resources	Scaling Policy
Staging	Intermediate environment for QA and UAT	us-west-1	1 replica	Auto-scaling
Production	Live environment for end-users	us-west-1	1 replica	Auto-scaling

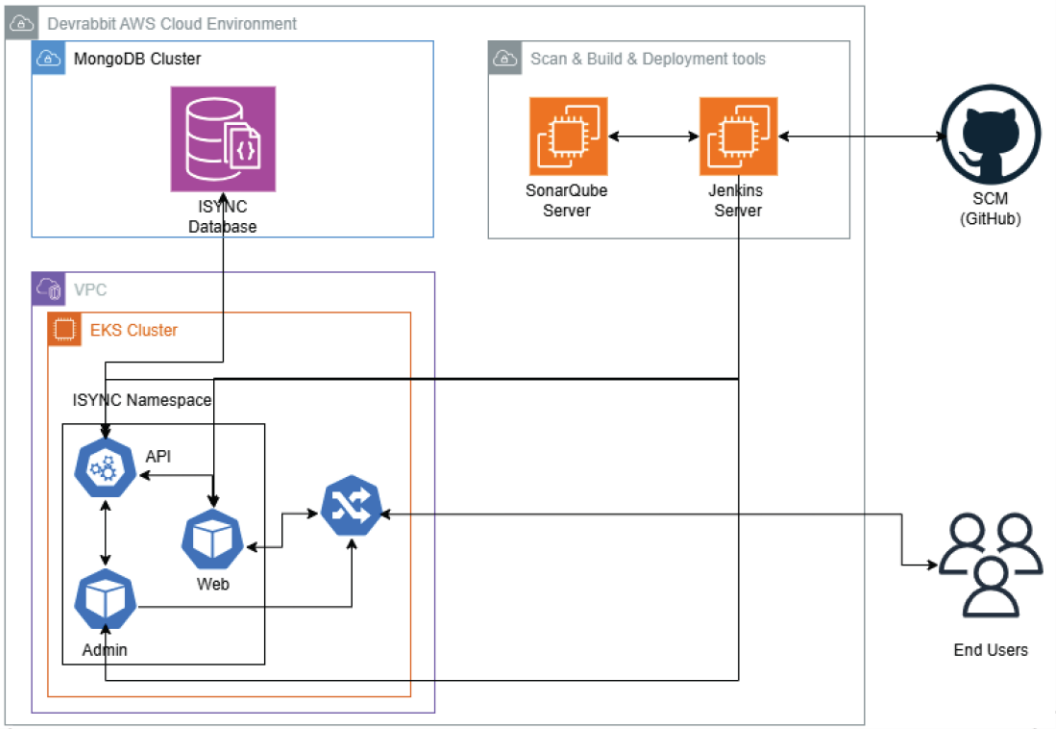
- **Auto-scaling**

  Implemented using Horizontal Pod Auto scaler (HPA) to ensure efficient resource utilization by automatically scaling the number of pods in response to real-time traffic spikes. HPA monitors key metrics such as CPU and memory usage (or custom application metrics), and adjusts the pod count dynamically to maintain optimal performance and cost-efficiency during high or fluctuating workloads.

- **Staging Environment**



- **Production Environment**



5. Deployment Process

5.1 Version Control

- Repository: GitHub
- All code is maintained in version-controlled repositories with separate branches for development, staging, and production.

5.2 CI/CD Pipeline

- Build Tool: Jenkins
- Quality Checks: Integrated with **SonarQube** for static code analysis.
- Artifact Management:
  - Docker images built on Jenkins are pushed to Amazon ECR.
  - Each build is tagged with the commit hash or version number.

5.3 Deployment Strategy

- Platform: Amazon EKS
- Deployment Method: Rolling Updates
  - Ensures zero-downtime deployments by incrementally replacing old pods with new ones.

5.4 Configuration and Secrets Management

- Environment Configuration:
  - Managed using Kubernetes, ConfigMaps and Secrets.
  - Secrets (Ex., DB passwords, API keys) are encrypted and injected into pods securely via service accounts.

6. Security Measures

6.1 IAM Role-Based Access Control (RBAC)

- IRSA (IAM Roles for Service Accounts):
  - Allows Kubernetes pods to access AWS resources securely using IAM roles.
  - S3 access, for instance, is restricted to specific service accounts only.

6.2 Network Security

- VPC Isolation: Applications run in private subnets.
- NLB (Network Load Balancer): Handles ingress traffic and routes it to the appropriate services.
- Security Groups & NACLs: Control access at subnet and instance levels.

7. Monitoring and Logging

7.1 Monitoring

- Tool: Grafana
  - Dashboards created for monitoring CPU, memory, disk, and network usage.
  - It is integrated with Kubernetes Metrics Server.
  - Alerts configured for threshold breaches (e.g., high memory usage, pod failures).

7.2 Logging

- Log Sources: Frontend, Admin, and API services.
- Log Aggregation: Logs are shipped to Grafana.
- Retention: Logs are retained for 7 days.
- Use Cases: Debugging, performance tuning, and audit tracking.

8. Use Case Elaboration

8.1. Use Case Scenario

A client has requested the integration of two independent systems — PlanGrid (as the source) and Nuvolo (as the destination). Below is a detailed overview of the end-to-end use case flow facilitated by the ISYNC application:

8.2 Use Case Flow

8.2.1 Client Request

The client approaches the team with a need to integrate PlanGrid and Nuvolo. This triggers the creation of a new integration opportunity.

8.2.2 System Configuration by Administrator

The admin sets up a brand new account as a customer, including all the required details, necessary data points, and integration limits.

8.2.3 User Registration & Integration Setup

The client logs into the ISYNC application and completes the 5 required steps using the integration wizard.

- ✓ Title & Description
- ✓ Source API Credentials
- ✓ Destination API Credentials
- ✓ Field Mappings (Destination to Source)
- ✓ Scheduler & Settings

8.2.4 Activation of Integration

- Once all steps are completed, the integration becomes active.
- Cron schedulers are triggered based on the selected frequency (e.g., every minute).

8.2.5 Automated Data Sync

At every scheduled interval:

- API calls fetch records from the source.
- Records are stored in the ISYNC database with status such as "initiated" or "update-request" based on uniqueness and status changes.
- Based on field mappings, objects are constructed and pushed to the destination system.

8.2.6 Handling of Create and Update Methods

- Records with create mappings are sent to the destination and stored post-push.
- Records with update mappings (matched via unique identifiers) are picked with status "update-request", aligned to mapping rules, and updated in the destination system.

8.3 Supporting Features

8.3.1 Exception Logs

All failed pull/push operations (due to data issues, API errors, etc.) are logged and available in the UI for troubleshooting and transparency.

8.3.2 Insights & Statistics

Graphs and counts of synced records, cron activity logs, and status summaries provide operational visibility within a selected date range.

8.3.3 Record Lifecycle Tracking

Users can trace each record's journey—from being pulled with a "New" status, transformed, pushed, updated, and evolving across different statuses like "checked out", etc.—with exact timestamps for each transition.