A picture containing text, indoor, shelf

Description automatically generated

**Catalyst**

Concept Note v1.1

[**Introduction 3**](#_e245j4e1j9da)

[**Problem Statement 3**](#_msbhbfs2rs31)

[Challenges 3](#_hbj5b42355lm)

[**Objectives 4**](#_krv63xg2lf3a)

[**Key Features 4**](#_a7n2b1jbno5q)

[**Architecture 5**](#_ktdyydkpj6lz)

[Infrastructure 5](#_kbya5zg71ff3)

[High-Level Design 5](#_swx0a2oat9pp)

[Datastore 7](#_7vg092ufq3wr)

[Production Readiness Checklist 7](#_jhlg48vnlg35)

[Impact and Sustainability <TODO> 8](#_1yftmed5wvc3)

[Supporting Documentation 8](#_oocx6ry1re0y)

## 

### 

## Introduction

**Catalyst [/ˈkatəlɪst/] noun**

* Any substance that increases the rate of a reaction without itself being consumed
* An event or person that causes great change

The purpose of the catalyst is to speed up the custom PG integrations between Razorpay and the merchant systems. The catalyst framework shall have all the building blocks which are required to onboard any merchant with Razorpay PG offerings

Repo: https://github.com/razorpay/catalyst

## **Problem Statement**

With our past experience working with NTRP, Rajasthan Govt eGras & IFHRMS projects, The Professional Services Team has noticed a few commonalities that can help us seize the opportunity in Govt & Banking verticals at scale. Following are the functionalities which are common in the aforementioned projects

* The information exchange between these systems & the PA/PG is always end-to-end encrypted
* They cannot integrate with the PA/PGs instead they expect us to integrate with their systems
* Typically these are S2S APIs & they refrain from calling multiple APIs/Endpoints
* All of the integrations follow the Browser redirect for Payment invocation
* The Govt entities will have more than 1 payment partner at their end, which makes it impossible to integrate Razorpay’s payment SDKs on their client apps

### Challenges

Currently, the PS team is facing challenges in scaling up the existing PHP-based service custom-solutions-web to meet the merchant requirements, here are a few reasons why:

* As an organization, Razorpay has decided to use Golang as a defacto framework for all backend services & maintaining a PHP-based service for newer implementation can increase the scope of migration in the future
* Enhancing the PHP code base to support Asynchronous processing on the workers is far more complex & redundant. The common utility package of Razorpay in Golang provides this out of the box
* The PHP service consumes more CPU & Memory as supposed to the Golang-based service, any enhancements here will considerably increase the overall infrastructure cost

## **Objectives**

* Migrate the existing solution from CSW to catalyst with a primary focus on optimizing the performance of crons and reducing CPU resource consumption.
  + This migration should lead to a more efficient and scalable system that can handle high workloads without causing significant spikes in CPU usage.
* Improve performance and resource utilization
* The transition from synchronous to asynchronous processing, where possible, to better handle long-running tasks or tasks with high concurrency requirements.
* Build a future-proof framework to take up all the custom implementations in the near future
* Leverage the best practices & common components built by other Engineering teams, keep the tech stack updated with org standards
* Last but not least, Build a reliable, reusable framework for the custom projects which can increase the Go-To-Market time for the new implementations by a minimum of 40% lesser ETA

## Key Features

* The key feature of Catalyst is faster processing without significant CPU processing. We will achieve this by using worker threads and Golang as our primary choice of programming language.
* Implement foundational design patterns for the use cases defined in the problem statement
* A common repository for all the custom reporting use-cases
* A common repository for all the custom webhook implementations
* Multi Data source connection will connect to the custom-solutions-web repository to support all the existing use-cases
* Support asynchronous processing using Kafka, 2-way pub-sub implementation between catalyst & custom-solutions-web
* Framework-level support to retries in case of external gateway API failures
* Out-of-the-box Mutex support for handling concurrency issues
* Ability to run crons directly from the Spinnaker pipelines, this can help the TAT for on-call issues

**Projects in pipeline for Catalyst**

* [**RBI Scroll - PACS 2.0 Solution Document.docx**](https://docs.google.com/document/d/1GyluKSIhQgkg-xAsxOChoX5TExV2Wnls/edit)
* [**E-Grass Scroll Files - BRD**](https://docs.google.com/document/d/1zatAobmjPixr1izMoK-CANtDufZgWSBcnFVagPGoB18/edit)
* [**NTRP PAO Integration - API Spec.doc**](https://docs.google.com/document/d/1zEg0YojKmBT-fJFl2L8Pej8RW8LF6D9b/edit#bookmark=id.gjdgxs)

## Architecture

### Infrastructure

Catalyst is a Golang-based microservice hosted on the Razorpay Non-CDE instance of the Kubernetes Cluster. The setup is similar to any other Golang-based service in the current infrastructure.

Some key considerations are given below

* Catalyst will not store card-sensitive data, hence it's a non-cde candidate.
* Leverage the domain, solutions.razorpay.com in production & solutions.stage.razorpay.in in the stage environment
* Route the traffic via the Edge gateway to leverage the rate limiter & other functionalities supported by Kong
* Setup web & worker pods on the Kubernetes
* Setup separate pipelines to execute the crons in Spinnaker
* Leverage existing static IPs already in place
* Connects to Razorpay’s Kafka Queues

Kubernetes configuration to begin with

Memory Allocation:

CPU Allocation:

Minimum Number of Web Pods/Cluster:

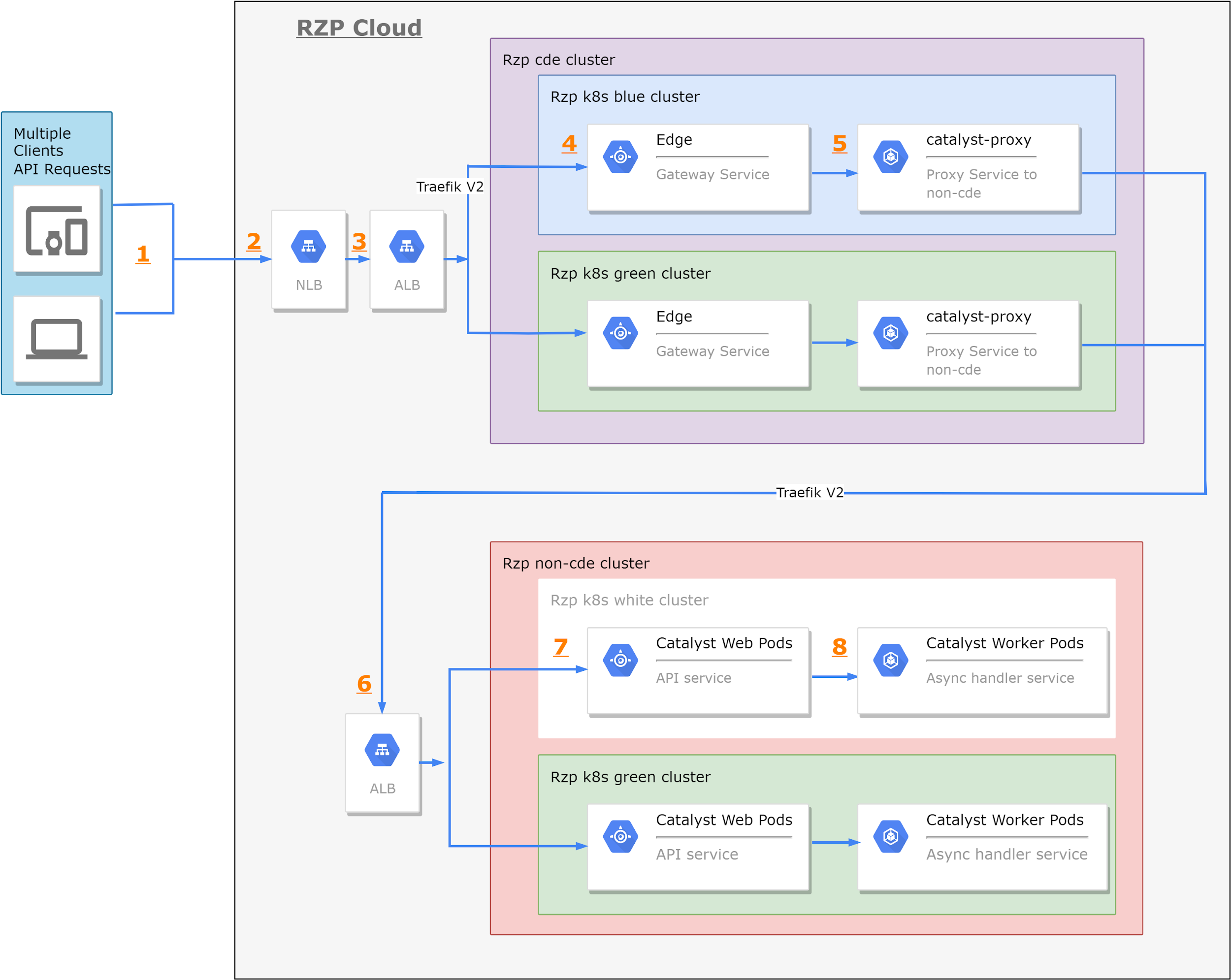
Minimum Number of Worker Pods/Cluster:

### High-Level Design

Basis the infrastructure considerations, here is the high-level architecture diagram of Catalyst.

**Pointers**:

* Razorpay’s infra has a CDE (Card Data Environment Zone) & a Non-CDE Zone & each Zone has 2 Availability Kubernetes clusters. Traffic between the clusters is distributed using an ALB
* Point 2 (in the diagram), the NLB is used to meet the static IP requirements of the NTRP project which is leveraged for the catalyst service as well
* As of date, Edge is available only in the CDE Zone of Razorpay, all the traffic of Non-CDE Zone services from Edge is routed through a proxy service (Point 5 in the diagram)
* All traffic pointing to the route solutions.razorpay.com/v2 is intended to reach the catalyst service in both stage & prod infrastructure.
  + Rules for routing the v2 traffic are added at the Edge Gateway  
      
      
    CSW - solutions.razorpay.com/status  
    Catalyst - solutions.razorpay.com/v2/check

[](https://app.diagrams.net/?page-id=f106602c-feb2-e66a-4537-3a34d633f6aa&scale=auto#G1fx1RF93qA9wLiDQRWVZS46hLAsfbQYiX)

Source: [Catalyst HLD](https://drive.google.com/file/d/1fx1RF93qA9wLiDQRWVZS46hLAsfbQYiX/view?usp=drive_link)

### Datastore

To begin with, the Catalyst service will have its own DB & the plan is to take every feature individually from the custom solutions web repo & migrate it to the Catalyst service.

Additionally, the Catalyst will have read access to the CSW Database for all the asynchronous/cron workloads.

The production cut-over will happen when the feature gets implemented in Catalyst & the database tables are completely in sync between CSW DB & Catalyst DB.

Keeping this in mind, the following are the considerations for the datastore

* SQL DB to support existing business use cases & for the newer implementations
  + Write Access to catalyst AWS Aurora instance
  + Read Access to CSW Aurora instance
* Redis for Mutex Implementation
  + Write access to the Catalyst Hosted Redis instance
* Kafka for the Worker Orchestration

An example of table for datastore consideration:  
  
 \DB::statement("CREATE TABLE `***garvi\_payments***` (

`**id**` char(14) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin NOT NULL,

`**merchant\_id**` char(14) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin NOT NULL,

`**payment\_id**` char(20) CHARACTER SET utf8mb4 COLLATE utf8mb4\_bin NOT NULL,

`**cin**` varchar(26) NOT NULL,

`**version**` varchar(5) NOT NULL COMMENT 'version column is for handling concurrency',

`**created\_at**` bigint NOT NULL,

`**updated\_at**` bigint NOT NULL,

primary key(id),

UNIQUE KEY `**payment\_id\_UNIQUE**` (`payment\_id`),

UNIQUE KEY `**garvi\_payments\_cin\_unique**` (`cin`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci;");

### Pod Architecture

The catalyst will be having 4 types of pod

* API - This pod will be the main pod which will receive and process the API requests. It will have a grpc server running at the port 8080, and an HTTP server which will be a proxy layer grpc server using grpc gateway on 8081. The metrics will be exposed at port 8082/metrics.
* Worker Pods - We will have multiple workers for different use cases. The workers will receive the jobs from Kafka and which worker will pick the task will be decided based on the topic of the Kafka message. The further details of the task will be in the payload of the message.
* Console - This pod will be the entry point for running all the crons. The Kubernetes will initiate a pod of this and run the command. Then depending on the command the console will push the relevant job to Kafka for workers to consume. This can also be done manually by running a spinnaker pipeline.
* Migration - This pod will only be used while running migrations on the database. We will be using the goose package to handle them.

### Production Readiness Checklist

* Dockerising the application -
  + Using the go-foundation for the boilerplate code
  + Docker & Harbor is supported Out of the box
    - Features: <https://wiki.rzp.io/DevOps/Application-Dockerizing-Checklist>
* Deployment
  + Stage Spinakker pipeline - ✅
  + Prod Spinakker pipeline -
* Monitoring
  + Stage Vajra dashboard - [Link](https://grafana.np.razorpay.in/d/73NnFyzSk/catalyst?orgId=1)
  + Prod Vajra dashboard -
* Alerts
  + Slack -
* Performance testing  
   K6 integration - L  
   P99, P95 and P90 metric - M
* Sanity
  + Probe check (/status) - Done
  + StatusCake integration - M
* Test coverage and automation - Harshil  
   Github actions for codeCoverage (unit tests >60%) - S - ✅  
   Sonar coverage (github action) - L - ✅  
   ITF support - S ✅
* Additional improvements:  
   Credstash integration - S  
   Devstack Integration - S  
   Graceful Shutdown (queue and workers) - M ✅  
   Edge onboarding - M  
   Auth and RBAC (for cron routes using SCJ). - L   
   Logging support to stdout - S ✅  
   Dark and canary pipelines - L  
   Asynchronous processing (using worker threads)- XL   
   DB selection (sql/nosql) - L ✅  
   Redis integration - M  
   HyperTrace Integration. - M   
   Splitz experiment support - M

## Impact and Sustainability <TODO>

* + Describe the expected impact and sustainability of the software product.
  + Identify any potential revenue streams or long-term growth opportunities.

## Supporting Documentation

* Health Server Tech Spec - [Catalyst Health Server](https://docs.google.com/document/d/1_2Oa0JjpX4N4drTr8KBoTCgbelepjofwT9sYjXXd5VM/edit)
* <https://alpha.razorpay.com/events/how-to-create-a-new-microservice-using-go-foundation>
* <https://www.youtube.com/playlist?list=PLy_6D98if3UJd5hxWNfAqKMr15HZqFnqf>