**HAPPIEST MINDS TECHNOLOGIES LTD.**

**PropertyGuru Section B – Technical Response**



**Copyright Information**

This document is the exclusive property of Happiest Minds Technologies Ltd. (“Happiest Minds”). The recipient agrees that they may not copy, transmit, use, or disclose the confidential and proprietary information in this document by any means without the expressed and written consent of Happiest Minds. By accepting a copy, the recipient agrees to adhere to these conditions to the confidentiality of Happiest Minds practices and procedures.

**Confidentiality Clause**

This document is being submitted to PG Enterprises Ltd by Happiest Minds Technologies Ltd. on the understanding that the contents of this document will not be divulged to any third party without the express written consent of the parties. It is also understood that the parties will not divulge any confidential information about PG that it may have access to during this interaction.

**Disclaimer**

This document has been prepared based on the information provided by PGL. Wherever proposed, the solutions and/or services mentioned are based on the requirements defined and understood by us at the time of preparing this document. While every effort has been made to make this document as accurate as possible, there might be changes to the document based on the subsequent discussions.

Table of Contents

[1. Executive Summary 5](#_Toc132708420)

[2 Requirement Details 5](#_Toc132708421)

[2.1 Functional Requirements 5](#_Toc132708422)

[2.2 Non-Functional Requirements 15](#_Toc132708423)

[2.3 In Scope 16](#_Toc132708424)

[2.4 Out of Scope 16](#_Toc132708425)

[2.5 Assumptions 16](#_Toc132708426)

[2.6 Dependencies on PG 17](#_Toc132708427)

[3 Technical Risks 17](#_Toc132708428)

[4 Solution Design And Approach 18](#_Toc132708429)

[4.1 Design considerations 18](#_Toc132708430)

[4.2 Modernization Strategy 19](#_Toc132708431)

[4.3 Layered Architecture 22](#_Toc132708432)

[5 Testing and Validation 24](#_Toc132708433)

[5.1 Baselining of Artifacts 24](#_Toc132708434)

[5.2 Test Requirement Analysis 25](#_Toc132708435)

[5.3 User Scenario Analysis 25](#_Toc132708436)

[5.4 Test Cases Design 25](#_Toc132708437)

[5.5 Type of testing 25](#_Toc132708438)

[5.6 UI & API Automation 32](#_Toc132708439)

[5.7 Test Types Vs Test Levels & Envs 32](#_Toc132708440)

[5.8 Test Execution & Defect Reporting 33](#_Toc132708441)

[6 DevOps Strategy 33](#_Toc132708442)

[6.1 Deployment Architecture for EKS 33](#_Toc132708443)

[6.2 Recommendation on Stack 35](#_Toc132708444)

[6.3 AWS Governance 36](#_Toc132708445)

[6.4 DevOps CI/CD Pipeline 37](#_Toc132708446)

[6.5 Branching strategy 39](#_Toc132708447)

[6.6 Release Management (support for versioning) 42](#_Toc132708448)

[6.7 DevOps Cloud Security for Application 42](#_Toc132708449)

[6.8 Disaster Recovery 42](#_Toc132708450)

[7 Technology Stack – Indicative 45](#_Toc132708451)

[8 Schedule and Milestones 45](#_Toc132708452)

# Executive Summary

PropertyGuru (PG) is Southeast Asia's #1 digital property marketplace, with leading positions in Singapore, Malaysia, Vietnam, and Thailand. PG currently hosts more than 3.5 million real estate listings, 40 million property seekers monthly with more than 64,000 active real estate agents across five growing economies of Southeast Asia. In 2021, PG completed the acquisition of iProperty Malaysia and ThinkofLiving.

PG has a single monolithic application called Guruland. This code serves all its marketplaces in Singapore, Thailand, Malaysia, and Indonesia. Guruland software was built with technologies that were in vogue in market at that time. Over time with expansion of business, new features were continuously developed and added to this monolithic codebase. PG intends to revamp its legacy system with suite of small and autonomous microservices. The project named “Regional Market Platform” is a concerted effort in the direction of rolling out the legacy system in a phased manner.

# Requirement Details

## Functional Requirements

List of microservices identified for migration to new architecture.

|  |  |
| --- | --- |
| **Service** | **Functions** |
| Listing Management Service | The objective of revamping this service is to build a new CRUD-based listing management service with a purpose-built data store, thereby omitting the dependency on PropertyDB.   * Post   + Select the tenant for which a Listing is being created   + Define the Listing details ( type of Listing, Listing Images, description, tenure, price range, valid until, landmark, size, add Agent Details etc. )   + Create tags for Listing - Like Built2018, Freehold * Get   + View All the created listing ( Active/ Expired)   + Paginated List, Search ( Listing ID ), and Filter ( date range, category, location, price range )   + Retrieve listing performance   + Retrieve listing rank   + List quality score   + Listing create date   + Listing expiry date   + Listing repost date   + Listing Views   + Listing impressions   + Listing Suspension - View suspension details, Appeal suspend listings, View Previous comments, Add comments   + Retrieve Performance report   + List property types   + List Boosted bookings   + List Past listings * Update   + Update any listing   + Update the attributes of a Listing like ( Image, contact details, property description, and tenure etc.)   + Repost listings - View Ad credits , Schedule Auto Repost listing   + Update listing status - mark rent/sold   + Reactivate suspend listings   + Update report listing rule * Delete   + Delete/Deactivate/Expire any Listing * Check PG Suggested Price   + Check price Suggestions from PG   + On click of this, PG suggested Price for the property should be displayed. The price should be as per the defined currency for the website.   + This helps in quoting the price for the listed Property * Toggle Feature   + Some functionalities are managed using feature-toggle, e.g. Tenant Based configuration( Make it live on selected site), Agent-Based configuration ( don’t show to specific agent group) * Content Validator   + To filter out unwanted language (racist keywords) from the listing details, content validator is used * Media Service   + Media related to listing is being delivered by the Media service (images, videos, floorplans etc) * Listing Performance   + Analytics around a specific listing is being handled by this service   + Listing performance overview   + Listing performance insights   + Listing competition summary   + Listing price insights * Listing Alert   + See when a New Listing is Posted , Deleted , Updated , Shortlisted * Listing statistics   + View listings data by date range and up to past 365 days * Shortlist listing   + See all the Shortlisted Listings [Wishlist kind of feature] * Recommendation   + Retrieve recommendation listing |
| Listing Search Service | Considering the difference in PG, iPP and BDS revamp this service to consolidate the search-related APIs to this unified service, including building a new API to pull expired listings from the master listing database. The main objective of having this service is to have the separation of concern between agent and consumer business.   * Search Listings based on role type   + Show Expired/ Live Listing / both   + Listing Result Details - Less info vs More info * Listing Search Use cases   + Suggestion based search   + Search based on location ( City, district, area)   + Search based on house Type   + Search based on Category Type [ Buy, Sell, Rent]   + Null Search   + Search based on Listing Attributes [No image search or voice search]   + Search based on Listing tag   + Search by MRT [train Stations]   + Search by HDB Estate   + Search by some Parameter   + Last searched Timestamp to be maintained   + Search based on type of transaction ( residential, commercial, land, agriculture) * List Search Result   + View Listing on Listing Result Page   + View Listing on Google Map   + Filter and Sorting options - Filter Listings by Recommended, Newest, Price, size)   + Save a Search result - Need login to save   + Display Agent Profile and Contact information with listing   + Youtube link for search Results   + Send Email to the Agent; opens in outlook (default)   + Whatsapp Integration with search result   + Hide or unhide any Listing   + Option to See and give Agent Reviews |
| Agent Subscription Service | A multi-tenant service which will host all the functionalities related to agent subscription, i.e., credits management, billing of agent products, and self-serve payments. It should support agent subscription-related workflows.   * Buy a package   + List packages - Standard, Advance, Premier, Business   + Accept/Reject quote   + List payment types - Full payment, pay-per-month, pay-quarterly, pay-half-yearly   + Make Payment - Third party payment integration * Renew a package   + List purchased packages   + List Package expiration date   + List quote   + Accept/Reject quote   + Make Payment - Third party payment integration * Upgrade a package   + List available upgrade packages   + List quote   + Accept/Reject quote   + Make Payment - Third party payment integration * Top Up Prime Credits   + Buy Prime Credits   + Make Payment - Third party payment integration * Top Up Ad credits   + Buy Ad Credits   + Make Payment - Third party payment integration * PG Money   + List Ad credits balance, expiry date   + Retrieve Ad Credits Consumption Report with filters   + Partner 360 Rewards tier   + List Package subscription   + List total number of Prime Credits   + List total number of Ad Credits * Ad Credits Spend Summary   + Views per Ad credit   + Leads per Ad credit   + Impressions per Ad credit   + Ad Credits Overview   + Itemized Ad spendings detail |
| Agent Management Service | A multi-tenant Agent Management Service that handles Agent domain for our marketplaces. It is expected to be created as a standalone microservice which will own the Agent and Agency domains. It will provide the APIs for the Agent UI service, i.e., acting as a backend for the frontend (“BFF”).   * Manage Agent   + Create Agent records   + List Agents   + Search Agents   + Filter Agents   + Update Agent records   + Send/ Resend Verification Email to Agent   + Send/Resend Activation SMS   + Retrieve Agent delayed membership   + Retrieve last 30 days stats of an Agent   + Suspend Agents   + Delete Agent record   + Sync user info to third party * Manage Agency   + Create Agency record   + Update Agency record   + Search Agency * Manage Company   + Update company   + List company by ID * Manage featured agents   + Search featured agents   + Expire featured agents   + List Featured agents * Manage Agent's Suspension   + List agent's suspension records   + Update appeal request   + Delete appeal request * Manage Users   + Create User   + Search Users   + Filter Users   + List Users * Manage User address   + Create User address   + Update user address   + Search user address records   + Delete an user address   + List user address * Manage Transactions   + Creates transaction record   + Update transaction record   + Search transactions   + List transactions * Manage role based access   + Create access rules   + Update access rules   + Delete access rules   + List access rule * Manage communication log   + Create communication log   + Update communication log   + Search communication log   + Delete communication log |
| Users Service | As part of this revamp, we want to make the user service a completely standalone service which owns the User domain completely. To achieve that, we would require to obsolete any dependency that this service has with legacy PropertyDB tables.   * Registration   + Role: Admin:   + CRUD ACL Rules   + Role: Agent/ Consumer   + Register as an individual or Company * Communication preferences   + Role: Consumers/ Agents / Admin   + CRUD Communication Preferences [ EMAIL, SMS, Whatsapp ] * Session management   + Configure Auto Signout time   + Send notification when logged in from another device   + Manage Session logs |
| Mobile BFF | A separated BFF layer for mobile devices is common for all our marketplaces. As part of the revamp of this service - we would focus on creating a BFF layer with exhaustive APIs which can power our mobile apps   * Property Agents API   + Sync user info   + Search agents   + Search by Agent id * Agent-website   + Update agent domain expiry date * Alert API   + Create alert   + Search for alerts   + Alert count   + Delete alert   + Retrieve alerts * Autocomplete search   + Autocomplete search for Property Listings * Autocomplete-agent   + Autocomplete search for Agent records * Boost-bookings   + Retrieve booked bookings (active and delay) by listing id   + Retrieve booking configuration * Mortgage calculator   + Saves calculated loan eligibility criteria of customer   + Invoke mortgage calculator widget * Developers   + Create developer account * dfp-targeting   + Google DFP targeting * Discretionary Product API   + Retrieve Agent specialists * Enquiry   + Create new enquiry for agent   + Create new enquiry record for the listing   + [Developer] Create new enquiry record for listing * Property Listing API   + Create new listing   + Search for listings   + List banned words/expressions   + Listing presence and validation rules   + Retrieve single listing   + Update listing with given ID   + Reactive expired listing - deduct credits   + Resume listing - Activate listing   + Update listing with given ID * Billing API   + Update start date of Agent * Map   + Map search for cluster of listings * Media API   + Retrieve image   + Upload an image   + Delete an media item   + Edit media attributes * Media – moderation   + Add media to moderation queue   + Retrieve moderation status of set of media   + Retrieve moderations status of set of listings   + Add moderation appeal for media * Package   + List packages for PG Money   + List summary of packages for PG Money * Price Index   + Search for price index * Products   + Create product * Properties   + Search for properties   + Price index for the listing/property   + Price index for District   + Search for property   + Search POI around property   + Price index for the Property   + Price index for property valuation   + Transaction for property * Purchased products   + Retrieve purchased count and utilization by user   + Update purchased product   + Activate purchased product   + Search purchased products by filters   + Expire purchased product by ID * Purchases   + Create Purchase   + Utilized Ad credits products * Reference   + Geographic reference data   + Reference data for an area/district   + Retrieve districts in a region   + Retrieve reference data for an area   + Retrieve Listing related reference data * Report   + Ad-credits reports   + Total Ad-credits report * Shortlists   + Add items to shortlist   + Search for shortlist items   + Delete shortlist item * Transactions   + Rentals and Sales data   + Rental and Sales data trends * User   + SignUp a user -Initiates sign up then redirects to propertyguru web * Utilized products   + Create a utilized product   + Deletes a utilized product * News API   + Retrieve news details based on parameters   + List all categories in a country   + Search News based on parameters   + Search News   + Retrieve list of news based on one or multiple tags * New Project reviews   + List reviews by language and region   + List reviews   + Terms Review * World data   + POI train stations from world sdk   + MRT stations * Property Developer   + Search for developer listings   + Count of developer listings   + List of featured listings by country   + Last video   + List videos   + Details of specific video   + New Home listing details   + List of Featured listings   + Search for overseas listings   + Count of world developer listings   + List of overseas features listings   + List of overseas new home listings |
| Admin Service (Frontend) | Create a micro-service which will cater to all the admin-related tasks such as impersonating an agent, managing agents, agencies, users, and other domain objects. In essence, it will be a single place from where we would be able to manipulate all the marketplaces within PG. We will also try to explore if the missing features from Salesforce could be added to this new service.   * Admin Service   + Act as Superuser to manage other users' and agents' accounts * Impersonating an agent   + Search for any Agent and look at the agent's profile, listing, and other details and use its account to perform functions like posting ads, replying to end customer etc. * Manage agents   + Active/ Inactive an Agent   + View Agent Profile Details   + Filter, search * Manage Agencies   + Active/ Inactive Agencies   + View all the agents under an agency   + Filter, search * Manage Users   + Active/ Inactive Users   + View User Profile Details   + Filter, Search |
| User Engagement – Enquiry | A multi-tenant service to handle all the leads and enquiries originating from any of the marketplaces. As part of the revamp, the dependency on legacy PropertyDB need to be removed, and a purpose-built database for this service is to be created. It will also make use of a messenger service to send different types of communications, i.e., email, SMS etc.   * Comment   + Send Comment for any selected Listing   + See list of Comments and Responses [ Reference Linkedin/ Teams kind of chat where all the chats are listed on the left ]   + Delete a Comment   + Update a Comment [edit] * Content Validator   + Ability to Report an offensive Content * Enquiry   + Send enquiry for any listing - Share Email ID, Phone number, and Name   + Enquiry Creates a Lead in Salesforce   + All enquiries go to an inbox [ under My Activity]   + The agent responds to the Enquiry * Feedback   + Provide Site Feedback   + Give your - Name, email , mobile, text and CAPTCHA |
| Advertising | A multi-tenant microservice to handle use cases related to featured agents or listings for all our marketplaces. To create this as a standalone service, we would need to remove the dependency from the legacy PropertyDB and migrate the tables to the purpose-built DB for this service.   * Advertising:   + Select Tenant(s)   + Option to create Advertisement for Agent/ Listing   + Option to Pull video/ image from media services   + Select the real estate [ Left Banner, Right banner, hero banner] and Name of the Page [ Listing Page, Detail Page ]   + Define the schedule of the Advertisement going live   + Ability to Cancel/ Stop or edit and update any advertisement before end time   + View Past Advertisement and filter by Category/tenant/date   + use old advertisement template and make edits to create a new Ad   + Use Ad Credits for posting advertisement * Custom Categories for advertisement   + Create Custom Categories for Advertisement [ On Sale, Easter Deals, Listings With good reviews, bad inventory etc]   + Select the tenant and Schedule the Launch time and date   + Select Listings for these categories - Search, Filter options |
| Developer (Database Dependencies) | The interaction of ProjectNet with PropertyDB needs to be omitted as part of this revamp. The dependent Tables need to be migrated to the respective services to ensure the services are bounded to their own domain objects. This will make ProjectNet a standalone service in the true sense.   * Project Net - PropertyDB dependency   + location   + media   + media\_item   + media\_original   + property   + property\_amenity   + property\_details   + property\_status\_history   + property (view) * ProjectNet – ETL   + streetnames   + location   + property   + property\_amenity   + property\_status\_history   + property\_details   + property\_type   + property\_unit |
| Info Service | A multi-tenant standalone microservice which will be responsible for handling all the requirements related to the price index, POI, transactions, valuations, property reviews etc.   * POI   + Point of Interest * Price Index   + Demand Index   + Price Index   + Property price index   + Subzone list demand * Price Index listing   + Retrieve Price Index * Properties   + Featured properties   + Search properties using free text and filters   + Property detail list, unit detail if floor and stack provided   + Property detail ID, unit detail if floor and stack provided   + Point of Interest near to property   + Price Index of property   + Rentals and Sales data of particular property   + Rentals and Sales data of near a particular property * Properties valuation   + Valuation of a particular property   + Information about regional transactions used for the valuation   + Information about property transactions used for the valuation   + Valuation of a particular unit * Property reviews   + List Property reviews   + Post a review for a particular property   + Details of property review   + Report a property review   + Submit a property review   + Update a property review   + Vote a property review * Territory pricing   + Local pricing information by district code   + Local pricing information by region code   + Local pricing information by territory code * Transactions   + Transaction data of sales or rentals for HDB or URA   + Rentals and Sales data   + Rentals and Sales trends data |
| Email Template Management | Move all the legacy email templates to the messenger service. The messenger service should be further enhanced to expose an interface where admins would have capabilities to manage different types of email message templates using WYSIWYG capabilities.   * View List of Email Templates   + View all the migrated multilingual templates based on region   + Upon Selecting a particular Template, the default email tool should open (e.g. outlook) * Create a New Email   + Ability to define a New Email template Category   + Rich Text Editor ( WYSIWYG capabilities ) to create a new template   + common header and footer and standard template structure for reusability & extensibility * View Past Emails   + View List of all the past Communications   + Ability to search or Filter based on category, Subject , Customer * Promotion   + Promotion and marketing emails |

## Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Services** | **Description** | **Remarks** |
| **Unit Testing** | Jest, Junit |  |
| **Containerization** | Docker |  |
| **Security** | Monitor & limit ingress traffic. | WAF- CloudFlare Web Application Firewall |
| Data encrypted in transit across the Internet and internally. | HTTPS |
| Data encrypted at rest. | Disk Encryption |
| Resources monitored for anomalous behavior and suspicious activity | Amazon GuardDuty |
| **Scalability** | Scalability (autoscaling) | ECS Autoscaling or EKS autoscaling |
| **Performance** | Latency is not degraded for a given functionality (UI/API) | PG will grant access to Datadog then add the load, performance, and scalability requirements |
| **Integration** | Percentage of successful integrations with the dependent systems |  |

## In Scope

* Decomposing existing Guruland monolith codebase into multitenant microservices. Changes are limited to PG in SG, TH, MY, and ID marketplace.
* Breaking existing monolith Guruland database schema to smaller schema aligning with the new microservices
* Data Migration from monolith system to microservices
* Web and App frontend integration with newly developed microservices
* Bifrost, Data Bridge, Salesforce, NetSuite, Workato, Segment and Braze changes limited to integration with new microservices.
* Functional and DB Testing
* API Performance Testing (for newly developed microservices)
* Testing Automation (API)
* Provide support for setup and automate the Infrastructure and CI/CD pipeline on AWS cloud for Dev, Testing, Staging and Prod environments.
* Knowledge Transition is limited to System Architecture, Database Schema and new microservices codebase.
* Produce wiki documentation for system design, sequence diagrams, deployment diagrams, and use case.
* Post-production 12 months warranty support (L3 Support)
* Support Security Testing (conducted by PG or identified 3rd party)
* All knowledge transition and technical documentation would be in English.

## Out of Scope

* Any services not mentioned in requirement section.
* Migration or decomposition of any service or codebase outside Guruland codebase
* iProperty (“iPP”) in Malaysia, Batdongsan (“BDS”) and thinkofliving in Vietnam Code/Service migration
* Mobile and Web UI new feature development
* Migration or changes to CMS layer, static pages including Agent offering and Area Insider applications.
* Change is flow for 3rd party Advertisements.
* UI/UX Design
* Post-production infrastructure support
* L1/L2 Support
* Security and PEN Testing
* User Interface internalization language Testing
* Product Documentation: FAQs, User manual

## Assumptions

***General***

* This effort estimation is based on the current understanding of the requirements and assumptions. The efforts might change based on the changes in scope or understanding of requirements in discovery phase.
* PG team will procure licenses for any commercial libraries used in the project.

***Technical***

* The happiest minds will baseline the Guruland codebase from feature, security, and performance perspective before the start of the development and will use the baseline as reference for developing new microservice. Any changes would impact on the project effort and schedule.
* PG will provide security and performance benchmarks for the new services.
* PG shall be taking care of any changes to be made in the external systems (Salesforce, Netsuite, Workato)
* As the existing tech stack is PHP Symfony and microservices will be developed using Springboot/Go, there would be no code reusability between the old monolithic and new microservice based applications.
* The test plan for existing features is present and would be shared with the partner.
* The solution and estimation are based on considering AWS cloud platform. The solution architecture and effort might change if a different cloud platform is chosen.
* PG to provide KT and access to existing Automation Framework and Testing tools.
* PG team will provide test data for executing migration scripts and performance testing
* Tenants will be managed in the existing Identity Provider service. Also, the user association with a tenant.
* Admin UI will be modified in Guruland to provision tenant selection in the UI for users having access to multiple tenants.
* The existing IDP service has support for Oauth2.0 to protect the new microservice endpoints.
* Existing email servers will be used for all the email notifications.
* Considering 16/5 (6:00 – 22:00 SGT) L3 support for all working days and on-call support for P0 issues during holidays and non -working hours

## Dependencies on PG

|  |  |
| --- | --- |
| **Dependency Factors** | **To be available by** |
| SMEs, Product owner of the application for requirement clarification, and review of the user stories | Start of the project |
| Technical SPOC for feedback and clarifications | Start of the project |
| Development Infrastructure Setup & Network Provisioning for development activities (on AWS or equivalent) | Start of the project |
| Knowledge transition of the existing architecture, codebase, schema, and features walkthrough | Discovery phase |
| Access to existing requirement documents, solution or design document, API documentation, DB schema, Test Strategy/Plan/Test cases, Automation scripts | Discovery phase |
| AWS account details | Start of the project |
| Licenses for commercial software’s | Start of the project |
| Email/Notification format and content | Start of the project |
| Code repository GitHub, Jira, SonarQube and Jenkins access. | Start of the project |
| Feedback for queries and clarification | With in 2 days |
| Feedback for milestone deliverables | With in 5 days |
| Review UAT Test Case document | As needed during the project |

# Technical Risks

|  |  |
| --- | --- |
| **Risk Item** | **Mitigation** |
| Enhancing Identify provider service to accommodate tenant management and associating user with tenants. | Analyze and plan the changes with appropriate prioritization of implementation tasks. |
| Data from multiple marketplaces will be migrated to new microservice to the tenant specific schema available in a single database. | A robust data migration process to be built to have hassle free migration, which handles all the error and retry cases. |
| Trigger failure during generation at source or while processing at the target. Huge overhead on database. | Avoid using triggers as much as possible. As a alternate update Guruland implementation to generate events from service layer. |
| Data from new microservice will be consumed over message events in monolith. To process the message, create new object relational models as existing in Monolith. A lot of business logic is involved in generating data integrated model before update. This leads to rigors testing of the existing system for regression. | A rigorous testing of the existing system for regression. |
| Integration with external systems such as Netsuite, Salesforce and Workato requires some configuration changes to consume the new endpoints with ability to identify whether to use the old or new endpoint. | Analyze and plan the changes with appropriate prioritization of implementation tasks. |
| Data integrity between Monolith and new microservices. | Periodic automated audit to check the correctness and integrity of data |
| A curved out microservice may slowly become another monolith. | Correct the design by splitting the service to multiple microservices. Make conscious design decisions not to augment a service with multiple functionalities. |

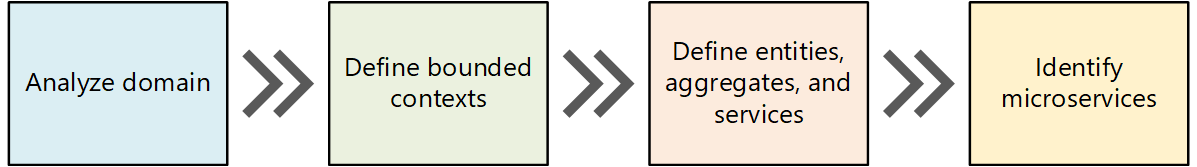
# Solution Design And Approach

## Design considerations

The proposed technical approaches for migration and To-be architecture after migration are based on the requirements specified in the RFP.

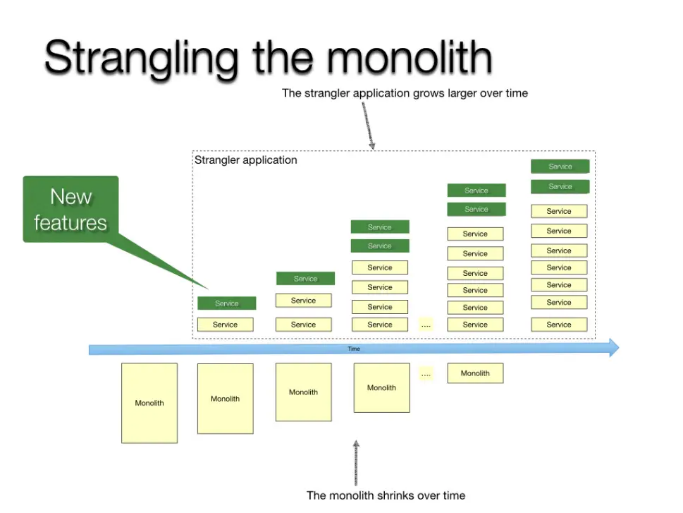
Following are the design considerations to Modernize PropertyGuru monolith application to microservices architecture:

1. **Domain driven design (DDD)** - Decomposition of PropertyGuru monolith application backend into microservices will be based on business domain as required. List of business services listed in section 2.1 are the subdomains or core domains which are considered for carving out the as individual microservices.
   1. Assumptions – We have assumed that Property Guru has followed DDD framework as below to arrive the candidate domains for microservices.



* 1. However, as we review the code and other artefacts we may add/modify services in consultation with PropertyGuru based on DDD principles and refine the overall microservices architecture. This may be due to a change in service boundaries, too many dependencies of components intertwined with each other etc.

1. **Business as usual** – Application migration will follow the [Strangler Fig](https://martinfowler.com/bliki/StranglerFigApplication.html) pattern to allow the migration of existing functionalities from monolith to microservices in an incremental manner, thus satisfying the requirement to continue the business as usual despite the changes to services. This approach will allow both monolith and microservices to exist together and provides a unified view of the application to the front-end layer until all services become independent. This enables us to avoid business disruptions and at the same time achieve migration in an incremental fashion as depicted in the below diagram.



1. **Multi-tenancy** - Each new microservice enables multi-tenancy by having tenant specific database connected through a single cluster of applications deployed as Software-As-A-Service model. The tenant specific configurations are pulled from configuration management system by deployment framework through Infrastructure-As-Code (IAC).
2. **Messaging and Integrations** – Establishing the APIs and deciding the best API communication pattern (asynchronous, synchronous, event driven) is key consideration for design and scalability of APIs. Events are generated to synchronize domain assets and their relationships `between monolith and new microservices. Event mechanism is leveraged to communicated between microservices to maintain the domain asset integrity.

* **Scalability** – Services will be scaled horizontally. The scaling factors will be based on inputs from the considerations such as observability metrics, performance metrics and business inputs. Current numbers with respect to the scale includes 3.5 million real estate listings, 40 million property seekers and 64000 active real estate agents.

1. **Security –** A layered security approach with controls at various levels such as network, api access level, user access, data security. Network segmentation, isolation of microservices to reduce blast radius, monitoring and logging, container security, Secure software development lifecycle addressing threat modeling of system are some of key design considerations.
2. **Resilience –** Automatic recovery from failure mechanisms (retries, circuit breakers, fallbacks), horizontal scaling, automating change managements and measuring against KPIs provided by PropertyGuru as inputs will be important considerations for the Resilience workload design.
3. **Data management considerations –** breaking, restructuring databases into single independent databases.

## Modernization Strategy

1. **Analyze & Identify**

In analysis phase, the domain specific individual services are identified based on the bounded context and those will be documented. The first service to be migrated will be decided based on the following factors.

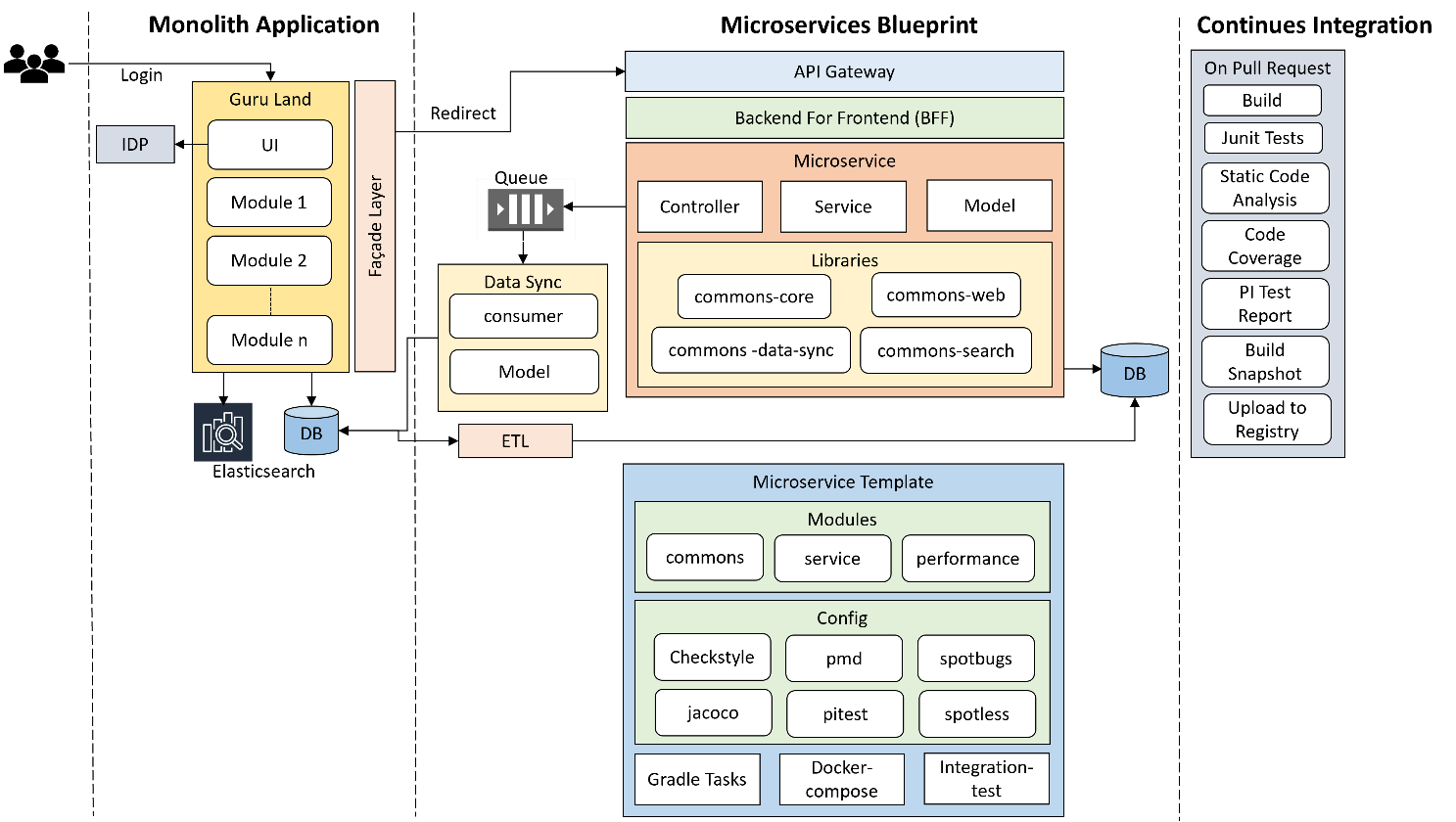
* 1. Domain which has moderate level of difficulty to handle with the monolith application.
  2. Reasonable level of risks calculated with mitigation.

1. **Transformation Blueprint**

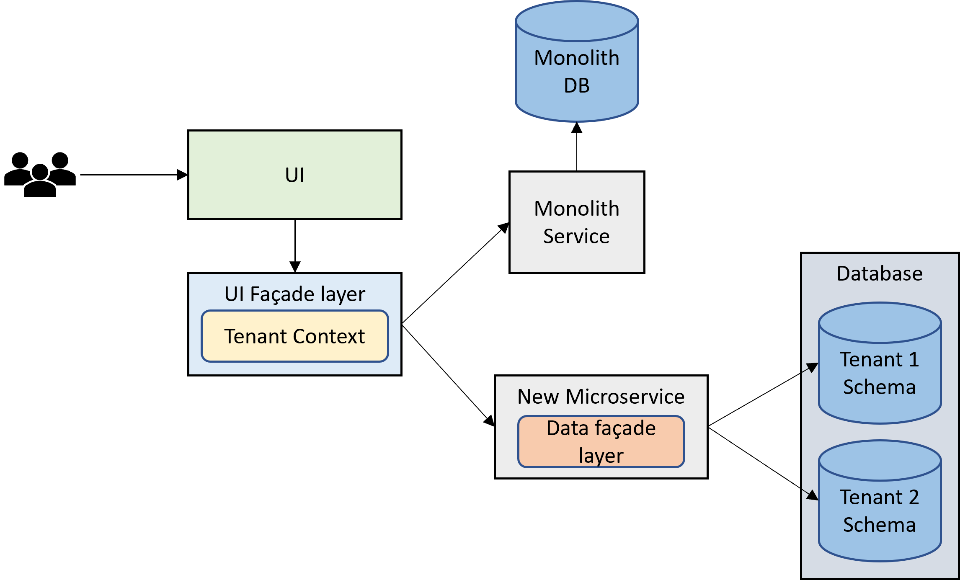
*Transformation considerations:*

* 1. Event driven architecture to generate events from the new microservice to propagate the data to monolith to maintain the data integrity of other domain features.
  2. ETL process to migrate old data of domain service specific schema. This ETL process to be created for each new domain specific microservices.
  3. Transactions in the old monolith application for the migrated domain generate events either from the service layer or as database triggers. Using these events, the new microservice data is maintained for integrity.
  4. Continue the user login from existing monolith and pass on the jwt token to the new microservice requests.

Prepare the development ground based on the Transformation Blueprint as defined below:



1. Define schema and model as documented during the analysis.
2. Define API contracts using OpenAPI Specification.
3. Create a microservice template such that the same template code is used to quickly bootstrap upcoming new microservices. This microservice template will be maintained as a separate repo in GIT.
4. Liquibase scripts to create and manage schema changes and data seeding.
5. Common libraries will be built incrementally by designing reusable components, utils. These common libraries will be added as dependency to the newly developed microservices.
6. Build infrastructure to generate events for the domain from the microservice.
7. A new microservice to consume, process and update the data in monolith system for these generated events. Message processing design to process events from different domain services.
8. Backend For Frontend (BFF) layer will be created to orchestrate the UI or Mobile client data contract across multiple microservices.
9. **Routing Request to new microservice**
   1. The communication between the presentation layer to service layer can be of two types:
      1. Through a call to the REST API served by API Gateway
      2. Making direct call to database through data layer components built in the monolith.
   2. A new façade layer will be built between the UI layer and API gateway by extending the monolith.
   3. On successful login the tenant specific information will be passed on as part of user request to create tenant context.
   4. In façade layer, create a tenant context with all the tenant configurations for each user session. It facilitates the routing to monolith data layer calls or to API gateway using the tenant context.
   5. Based on the tenant context route the downstream call to either to new service endpoint or to the existing flow or endpoint.
   6. Routing at the API gateway will be done based on the API URI defined for old API and new API. The tenant information will be propagated to new microservices over request headers.
10. **Multi-Tenancy**
    1. Single application supporting multiple marketplaces as multiple tenants.
    2. Each microservice will have a separate database. A separate schema will be managed for each tenant.
    3. A data façade layer will be created as part of the microservices foundation to identify the tenant specific datastore to connect to the tenant schema.
    4. The new façade layer created in the monolith will identify and create the tenant content.
    5. The tenant context will be propagated to the microservices using which the tenant resources will be identified.



1. **Data integrity and rollback.**
   1. The data of the migrated domain will be managed in the new microservice schema. And the other domain data will be continued to manage in monolith.
   2. The integrity of data between the new microservice and monolith will be maintained by syncing the data between monolith and new microservice through event mechanism.
   3. Any point in time. the new microservice can be rolled back or plugged off from the ecosystem. This will not impact on the business.
   4. Each new micro service creation will have separate migration scripts to migrate data from monolith for the specified domain.
   5. Data from multiple marketplaces will be migrated to new microservice to the tenant specific schema. A robust data migration process will be built to have hassle free migration.
2. **Data Sync from new microservice to monolith**
   1. A new Data sync microservice will be created to manage the event generation and event processing infrastructure.
   2. Tenant specific events are generated and consumed over message queue.
   3. Separate message consumers are created for each microservice to update the data in the monolith.
   4. New model hierarchy as per the monolith object relations will be created to process and update the data in monolith.
   5. A lot of business logic is involved in generating data integrated model before update.
   6. A robust message exchange infrastructure will be built to handle the failovers to maintain the data integrity at the monolith.
3. **Microservice Template**

Microservice template will have the following things added incrementally during the development of the first microservice.

* 1. Gradle build files with settings, plugins, dependencies, and tasks to build, assemble, verify, and test.
  2. Configuration for code quality tools such as Spotless, Checkstyle, PMD, Spotbugs, Jacoco, PI test. These tools will be executed on Verify Gradle task. The same will be configured in the configuration management tool on raise of a PR.
  3. Docker-compose and Dockerfile having the required tools to help create the container for local testing.
  4. Basic infrastructure to read configurations for external systems like Database, Cache, Message Queues etc.
  5. Standard modules such as common, service, and perf with Foo domain implementation of Controller, Service and Dao classes. Junit test cases for these Foo implementations.

1. **CI on Pull Request**
   1. Every Pull Request of the development requires a set of tasks to be performed as quality gates before preparing the deployment artifact.
   2. Development configurations and Gradle tasks will be prepared for quality gates execution.
   3. CI initiates the build, unit tests, integration tests, Static Code Analysis, Code Coverage, PI tests as part of the quality gate.
   4. The artifact generated post compliance of all the quality checks will be uploaded to Container Registry.
   5. CD scripts prepared will be used to deploy the artifacts on to the respective environment.

## Layered Architecture

High level System Architecture (To-be architecture)

A screenshot of a computer

Description automatically generated with medium confidence

The system will be organized in a layered approach for clear separation of responsibilities. The layered approach promotes re-usability, scalability, maintainability, and fault tolerance as layers/components have loosely coupled with each other. The advantage being, that the layers can be implemented, tested, and deployed independently.

***API Layer***

* Platform features are decomposed into multiple microservices which can developed, tested, and deployed independently.
* There will be an API Gateway for all the backend services which are mainly consumed by Web client applications.
* All tailor-made composite APIs for the business requirements will be exposed through API Gateway.
* Platform core will be built to manage the microservice foundation services such as Security, Data Access, Caching, Error handling and Logging.
* Application Security
  + API will be secured through API Key, which the Web clients will pass over auth headers on each request.
  + API will be authorized using auth tokens for each user based on their roles and permissions.
  + OWASP Top 10 security flaws and vulnerabilities will be addressed through static code analysis.
  + Application Security will be based on OTP send through SMS or email.
  + All communications are over SSL, which secures the request-response with data encryption.

***Microservices***

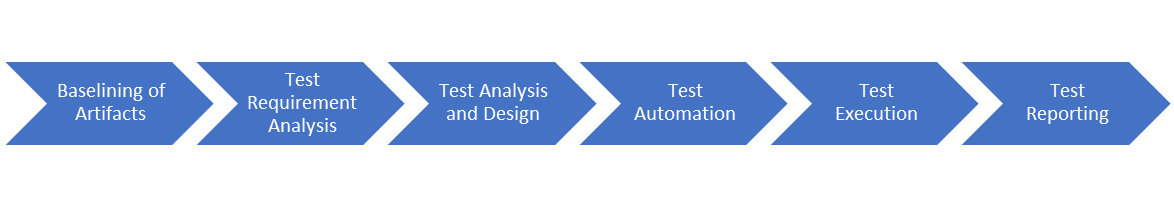
All the domain features will be implemented using micro-services architecture principles. This would mean all domain functions will be built as a self-contained service, which will be responsible for addressing the concerns for that specific domain functions.

***Data Layer***

* Depending on the Data modeling, flexible Schema, ease of querying, analytical and legacy data base bindings we will either use SQL based Relational Database services or NOSQL data Storage such as MongoDB.
* SQL
  + Relational database is used as transactional data store.
  + Independent schemas will be created for each microservice.
  + Legacy data storage and ease of querying requirements
  + Services requiring strong relational data model.
  + Database instances with support for fail-over through master-slave mode.
* MongoDB
  + Extensible schema, flexible schema, and horizontal scaling requirements.
  + Service requiring purpose-built requirements such as Document storage.
* File Store on cloud and Local file server
  + Cloud storage will be used to store all the archive data. Cloud storage has capabilities such as scalability, data availability, security, and performance.
* Cache
  + Frequently accessed data will be stored in an in-memory storage, such that the data can be served faster, by cutting down the database server round trips. Especially, messages and at its attributes will be cached for faster retrieval.
  + Redis Cache shall be used as in memory database storing various data structures such as Strings, hashes, lists, sets, and key-value pairs.

# Testing and Validation

Following lifecycle shall be followed for Quality Assurance in the E2E QA Team



## Baselining of Artifacts

Team shall baseline all the references and the available documentation along with brainstorming with the customer stakeholders to ensure all available artefacts are baselined at the start of the project.

In case of AGILE projects where the architecture and designs evolve, necessary processes definitions shall be identified and concluded during this phase so that everyone in the team is clear on what is expected at each phase of the SDLC.

Happiest Minds’ Simple Design guidelines shall be followed in these cases.

## Test Requirement Analysis

* Happiest Minds will document Test scenarios and perform Functional Testing, Integration Testing and End to End Testing that has been described in scope of work. Following is the approach which will be taken for functional testing of the system.
* Testing will be performed on the below PG App for the following applications,

1. Including iProperty (“iPP”) in Malaysia
2. Batdongsan (“BDS”) in Vietnam

## User Scenario Analysis

* We would be analysing various user scenarios based on how various roles who would be using the features as mentioned in Requirements sections for Listing Management Service , Listing Search , Agent Subscription , Agent Management, Users Service, Mobile BFF, Admin Service (Frontend) User Engagement – Enquiry , Advertising , Developer (Database Dependencies) , Info Service Email Template Management , and use the existing end-to-end test scenarios as part of good test coverage. Testing the application as per business logic and with focus on end user perspective would be considered.

## Test Cases Design

* ***New features/requirements:*** we would be documenting detailed test cases to be used for execution using the Test Scenarios identified from the previous steps. Test cases would be prioritized as P1, P2 based on the criticality of the test case.
  + P1 test cases include all the Critical requirement of a particular feature/requirement.
  + P2 test cases include major functional part as well as negative flows to validate behaviour and error/warning messages thrown by the application
* ***Regression features/requirements:*** Existing feature API Testcases would be provided by PG Team and the new enhancement features API testing would be taken care by HappiestMinds Team. In case testcases are not available only high-level test cases would be documented to cover the existing features to ensure new feature implementation has not introduced any defects in the application.

## Type of testing

Below mentioned are the various testing types that would be considered during testcase design.

|  |  |
| --- | --- |
| **Test Methodology** | **Description** |
| UI /Functional Testing | The functional testing shall be carried out as per the identified requirements for Listing Management Service ,Listing Search ,Agent Subscription ,Agent Management ,Users Service ,User Engagement - Enquiry ,Advertising ,Info Service ,Email Management in the requirement documents / brainstorming with the engineering teams and product managers / standards that are planned to be adhered to.In areas where there are enhancements / feature additions / refactoring planned on the core components, QA will study the as-is behavior in any of the existing systems and ensure the as-is functionality is intact wherever the requirements are ported without changes from as-is (Existing) to to-be (new implementation) systems. For areas where there are changes (Modify & Delete) to the existing functionality, QA will study the impact of the change and identify test scenarios accordingly |
| AB Testing | Validation will be done using Sixpack testing framework provided by PG, QA team will Install Sixpack, Set up our experiment and its variations, including the metrics needs to be measured, Add the necessary code to PG website or application to track conversions and send data to Sixpack and execute it . Analyze the results using Sixpack's built-in reporting tools and which variation performed better and provide statistical significance testing.  QA will validate various PropertyGuru new and old application scenarios that can help improve the user experience and increase conversion rates.  Testing Different PropertyGuru Homepage Layouts: Testing different homepage layouts to determine which design drives more user engagement and conversions.  Testing Search Filters of PropertyGuru: validate different search filters to determine which ones lead to more successful searches and property inquiries.  Testing PropertyGuru Details Page Layout: Testing the layout and format of the property details page, testing whether a property video increases engagement compared to a photo slideshow.  Testing Lead Capture Forms: Testing the length and content of lead capture forms can help determine the optimal number of fields and types of questions to increase the likelihood of user conversion.  Testing User Experience: Testing different user experience features, such as chatbots or property comparison tools to improve overall user experience. |
| BFF Mobile Application Testing | Validate the integration of Mobile BFF & Agentnet Apps with new corresponding micro- services .QA will validate the functionality of mobile application on latest IOS and Android devices.  All API’s available for mobile application would be automated as part of API automation task |
| API Testing | Testing the existing features and testing the In scoped features.  API Testing would be performed on Swagger / Postman as suggested by PropertyGuru.  QA Team shall identify all the APIs (Internal & Third Party) and identify all the changes in terms if Request and Response Data Parameters, HTTP Header changes for encoding etc. and Identify the test cases to be verified from existing set of API Test Cases from PropertyGuru  All Application Programming Interfaces exposed as a part of the system to the external or within the subsystem will be subjected to validation for response. REST API testing is performed at a message layer. API testing is used to determine whether APIs return the correct response (in the expected format) for a broad range of feasible requests and also react properly to edge cases such as failures and unexpected/extreme inputs. Test case design for API testing will be based on   * Return value - Return value based on input condition * Does not return anything - When there is no return value, behaviour of API on the system to be checked * Trigger some other API/event/interrupt - If output of an API triggers some event or interrupt, then those events and interrupt listeners should be tracked * Update data structure - Updating data structure will have some outcome or effect on the system, and that should be authenticated * Modify certain resources - If API call modifies some resources then it should be validated by accessing respective resources   API testing should be first type of testing that should be considered for Test Automation as most of the other sub system and the application working depends on the response of the exposed APIs |
| Integration Testing | Platform Integration points for each of the components will be identified and the test scenarios for the Integration testing will be designed. Main focus will be validating the component ability to integrate into the complete Platform and adherence to the data flow and the control flow paths of the Platform as designed for PropertyGuru. |
| DB Testing | PG Database validation – QA will validate the integrity, accuracy, and consistency of data during the migration process.  PG Data completeness: QA will ensure that all the necessary data is migrated to the new database, and no data is lost during the migration process.  PG Data accuracy & Consistency: QA will ensure the data is accurate, and there are no errors or inconsistencies in the data and ensure that the data is consistent with the original source, and there are no discrepancies between the original data and the migrated data.  PG migrate with New microservice: QA will ensure that each microservice is migrated with individual database created and validate the accuracy and consistency of data.  Data integrity: Ensure that the data is complete, accurate, and consistent, and there are no violations of data integrity rules, such as constraints, foreign keys, or unique indexes.  Performance & security validation: Ensure that the migrated database performs as expected, and there are no performance issues that may affect the overall performance of the application and ensure that the migrated database is secure and meets all the security requirements, such as access controls, authentication, and authorization.  Data validation: QA will ensure that the migrated data is validated against business rules, and there are no data quality issues that may impact the business processes. |
| Migration Testing | System testing will be performed in this case with all the data, which are used in PG legacy application, and the new data as well. Existing functionality will be verified along with the new/modified functionality of PG.  Migration testing will include testing with old PG data, new data, or a combination of both, old features (unchanged features), and the new features, Different phases of the Migration test which will be performed for PG applications  **Pre-Migration Testing**  QA team will set testing activities which would be performed as a part of the Pre-Migration test after discussion with concern stakeholders.  QA team will Perform data mapping between legacy and the new application – for each type of data in the legacy application compare its relevant type in the new application and then map them – Higher level mapping.  QA team will execute test cases, test scenarios, and use cases for new conditions in the new applications.  **Migration Testing**  QA team will validate PG application meets the expectations of the end-users after the migration process.  QA team will validate the user experience is not affected after the migration process.  QA team will validate all functionality and features are working as expected after the migration process to the new/existing users.  QA team will validate all data has been migrated accurately from the old PG database to the new one.  **Post Migration Testing**  QA team will validate PG application end-to-end system testing is performed in the testing environment. All relevant identified test cases, test scenarios, use cases with PG legacy data as well as a new set of data.  Backward Compatibility Verification  QA team will validate the new PG system supports the functionality supported in earlier versions along with the new one.  **Rollback Testing**  Migration failure at any point of time during migration, system will be roll back to the legacy system and resume its function quickly without impacting the users and the functionality supported earlier  QA team will validate application is able to rollback database changes made during the migration process to the previous state.  QA team will validate application is able to handle any issues related to data loss or corruption during the rollback process.  QA team will validate the rollback testing scenarios help to ensure that the application is able to rollback successfully in case of any issues during the migration process and recover to the previous state without any negative impact on the application's functionality, performance, or user experience. |
| Regression Testing | Regression testing will be performed before each milestone release by selecting and executing relevant test cases. Selective regression will be done as per impact of defect fixing or new module integration. Complete regression will be done during end-to-end system testing. |
| Performance Testing | Performance of the application/system/APIs decomposed out of PG should be similar and improved when compared to the corresponding monolith implementation.  The performance testing will be conducted according to the performance benchmarks (existing performance data will be provided by the PG team during the discovery phase) and will be used as the benchmark and come up with performance statergy accordingly. Performance team will ensure that the application does not degrade in performance compared to existing data.  Performance Testing shall focus on the following aspects Load Testing The entire solution is tested against the expected normal/peak load conditions (by simulating expected concurrent user loads & workload patterns) to verify whether the application can meet desired performance objectives.  This is done on a single instance without any load balancing.  Objective is to identify the peak load; one instance of the system can take. This can be a vital input for capacity planning of the infrastructure Payload Testing The system is tested against the data which is ingested with large payloads based on the identified boundaries. The payload is passed with maximum length for each of the parameters that the application accepts. Stress Testing The entire solution is tested with gradually increasing load until it reaches its breaking/saturation point in terms of response time, throughput & system resource utilization levels. This is done on a single instance without any load balancing.  Objective is to identify the peak load; one instance of the system can take. This can be a vital input for capacity planning of the infrastructure Endurance Testing Subjecting the application to expected user load over a long period of time (8 Hrs, 12 Hrs, 24 Hrs etc.). Focused on validating the performance characteristics of the application when subjected to normal production workload over an extended period of time and coming up with the system maintenance plan.  This is done on a single instance without any load balancing during Functional and Performance testing and once on full setup (including the final application) as part of E2E Testing. Scalability testing This is to determine the user limit for the application and ensure end user experience, under a high load depending on the application that is being tested, with different parameters: CPU usage, network usage or user experience. Reliability Testing This section covers the approach towards confirming the service continuity of the solution during local failures or during disasters. It also confirms the functionality of that portion of the PG system which considers failsafe arrangement to ensure a) service continuity during system failure and b) load-balancing for optional utilization of infrastructure components. System availability is calculated by MTTF (mean time to failure) and MTBF (Mean time between failures).  Execution shall be carried out in Multiple Test Cycles incrementally. The Application Under Test (AUT) shall be loaded in steps (from 10% to 100%) and a full Automation Test Cycle shall be run to monitor the system behavior. The sequence shall be repeated with multiple loads. Backup Testing The system is tested to validate its ability to handle backup of different configuration, data, and state information on redundant systems to that they can be recovered when the system crashes or there is an ungraceful exit  **Recovery or Failover Testing**  The system is tested to validate its ability to handle the load when one of the server/components fails  It shall also focus on ensuring there are no increase or decrease in the resource utilization (CPU, Memory etc.) when the system recovers  There will be additional tests planned on the client and server side to measure the Throughput and Response times of all the content and pages for varied concurrency. |
| Security Testing | Identification of Trust Boundaries Detailed Analysis shall be done and the trust boundaries at the sub system and the overall solution level Threat Modelling & STRIDE Analysis This shall be performed at every sub system level and trust boundaries will be defined at the overall system level and sub system level. Each of the System / Subsystem level interfaces shall be analyzed for Spoofing, Tempering, Repudiation, Information Disclosure, Denial Of Service and Elevation of Privileges. Based on this all the applicable threat vectors and the related security test scenarios shall be identified Infrastructure Scans & System Foot-printing Overall network topology shall be analyzed for Hardware, Firmware, Software Versions and any existing vulnerabilities are present in those.  Security w.r.t Data in Processing, Data in Transit and Data at Rest, Encryption at the database and logs level and access and privileges to different admins and users of each of the portals shall be validated. Security Audit and Scans Infrastructure and Static Code Scans shall be done using the Commercial & Open-Source Tools and Happiest Minds team shall present Scan Reports. Suggested scans: NESSUS, NMAP, FORTIFY (Code Scans), Application Vulnerability Scans etc. Packaging & Deployment  * Anti-Virus Scans for Packages developed and transferred to PG Infrastructure and Deployment team. * Validation of Undisclosed Interfaces, Backdoors and User sensitive Data Leaks * Validation of Standard encryption mechanisms, Key Management * Tracking of Software Deliveries over Secured Channels  Application Vulnerability Assessments  * Validation of Data Coupling and Control Coupling across interfaces for preventing DOS issues * Remediation Support and recommendation * Automated & Manual testing of application * Tracking of Protocol, OS, Servers, Open-Source Versions and other infrastructure Vulnerabilities * Validation of the behavior of the system for top OWASP vulnerabilities. |
| Microservice Validation | Functional Testing: Each microservice's when developed, QA team would perform basic functional testing and cover all dependent scenarios for the specific microservice unit.  QA will cover all functionality by creating test cases that verify that the service performs as expected. This includes testing the API endpoints, input validation, data storage, and data retrieval.  Integration Testing: QA will validate the microservice's integration with other microservices, APIs, and systems. This includes testing the microservice's ability to communicate with other services and its ability to handle exceptions.  Performance Testing: QA will test microservice's performance by simulating a realistic load on the service and measuring its response time, throughput, and scalability.  Security Testing: QA will test microservice's security by identifying and fixing vulnerabilities, such as injection attacks, authentication and authorization issues, and data privacy violations.  Resilience Testing: QA will validate microservice's resilience by simulating failures and testing its ability to recover from them. This includes testing the microservice's ability to handle network outages, resource shortages, and other unexpected events.  QA will ensure that your microservice meets the necessary requirements and standards and delivers high-quality results. |

## UI & API Automation

For the Identify phase to start, it is very important to have complete data on the test cases and test data. With no formal test cases being present, it would be ideal to have the one round of manual testing to be performed on the system before the start of the identify phase. During this period, focus can be on:

* Identification of Sanity test cases which needs to be automated as part of UI automation.
* Automation feasibility Study
* ROI identification - GO or NO-GO decision.

Post this, subsequent phased will focus on define the tools, framework, design and optimization. Plan can to complete the Identify, define and the design by end of first Phase.

Ideal approach of the test case finalization for Automation would be in the order mentioned below:

* API automation

Automated API tests provide much quicker test results and significantly accelerate development workflows; thus, it helps us speed up the feedback loop and catch issues faster. API automation will be implemented in a way in POSTMAN such that all the tests can be integrated with CI / CD Pipelines.

* UAT Test case
* Sanity test cases
* Functional & Non-functional test cases that become the part of the regression test suite.
* Compatibility test cases

## Test Types Vs Test Levels & Environments

Test levels and environments will be determined for each test type of the Application / Subsystem. Each Sub System level test strategy will incorporate this information.

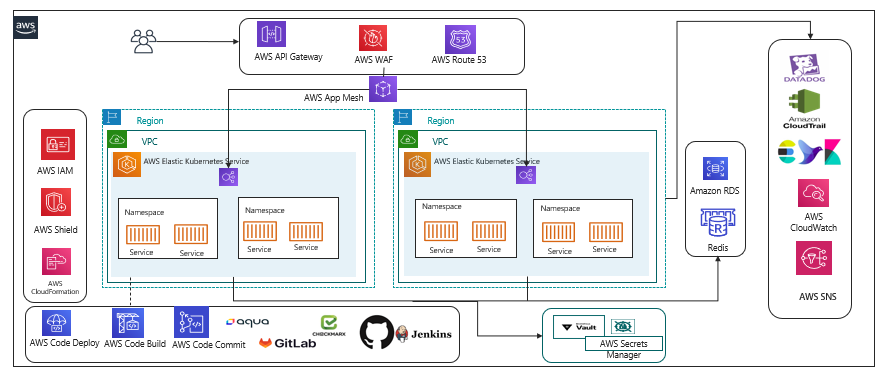
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Dev Environment** | **QA Environment** | **Bug Fixes** | **Before UAT** | **Pre- Production** | **Production** |
| Sprint Level | (Story Level) | (Story Level) |  |  |  |  |
| Rest API's |  | Regression | Sanity | Regression |  |  |
| Web portal |  | Regression | Sanity | Regression | Sanity | Sanity |
| Mobile application |  | Regression | Sanity | Regression | Sanity | Sanity |
| Integration Testing |  | Regression | Sanity | Regression |  |  |
| Performance Testing |  |  |  | Yes |  |  |
| Security Testing |  |  |  | Yes |  |  |
| Field Testing |  |  |  |  | Sanity | Sanity |

## Test Execution & Defect Reporting

* ***New features/requirements testing:***  Test cases would be executed on the dedicated test environment and devices. Defects would be categorized as Sev1, Sev2 & Sev3 and would be documented in the agreed upon defect tracking tool or excel document. (Sev 1 would be like application crash, functionality itself is not working. Sev2 would be functionality has issue but there is a work around. Sev3 would be cosmetic defects like UI issues, error messages are not informative, grammatical errors etc...).
* ***Regression features/requirements testing:*** Regression testing would be executed as per the test cases provided by PG Team or documented by Happiest minds.

# DevOps Strategy tr

## Deployment Architecture for EKS



**EKS Cluster**

Cluster is being enabled with the EKS managed cluster which will be distributed across the regions for the deployment so that the distribution of the load as well as the High Availability is being made available.

The Clusters are being configured in two regions for the High Availability

**Elastic Container Registry**

The base images are created for the containers to be deployed on EKS. We need the centralized storage for configure and uploading the non-prod and prod environments.

**Terraform**

Terraform is being proposed for the IaaS and CaaS.

**AWS Services**

AWS Cloud Formation template will also be used for configuring and provisioning the instances.

**Helm**

Release management is being handled by Helm Templates

Helm a package management tool will be configured for the release process and with the version control enabled.

The namespaces are being configured for each project deployment.

**AWS App Mesh**

App Mesh removes the need to update application code to change how monitoring data is collected or traffic is routed between services. App Mesh configures each service to export monitoring data and implements consistent communications control logic across your application.

This makes it easy to quickly pinpoint the exact location of errors and automatically re-route network traffic when there are failures or when code changes need to be deployed.

App Mesh uses the open-source Envoy proxy, making it compatible with a wide range of AWS partner and open source tools

**Aqua:**

Aqua will be integrated as part of the platform to validate the image being uploaded on to the repository.

Aqua is integrated with the pipeline to validate the security check for the vulnerability in the images

Security checks on the containers, images, code quality, integration of the vulnerability check will be integrated as part of the pipelines

**Application Monitoring :**

EFK: The application log monitoring is being taken care with the integration of the EFK (Elastic Search FluentD & Kibana)

This is centralized and configured in the regions for extracting and maintain the logs.

Additional Disk Volumes will be mounted as part of it so that the logs are always dumped and can be extended without the changes affecting the system.

The application and infrastructure monitoring are being monitored using the CloudWatch/Datadog.

**Infrastructure Monitoring**

The infrastructure are monitored using the aws managed service Cloudwatch or Datadog (only if license is available).

**Datadog Metrics Monitoring**

Datadog provides the APM tools for monitoring the traces of the applications and also the database time execution.

Datadog is tightly integrated with the Kubernetes pod for the monitoring of the internal execution along with the backend connection and the traces of the database execution.

Performance and the time taken for the execution of the queries are being traced and visualized at the centralized dashboard for monitoring.

**CloudTrail:**

Enables the auditing and monitoring API calls made with the AWS Accounts.

**Vault / Secret Management**

Secret management stores all the secrets and the cert keys, the projects will fetch the secrets from the management for the respective project.

**Security Protections**

AWS Shield for the protection across the DDos attack.

**Redis Cache**

**Redis caching for the faster process on the Database and the app**

## Recommendation on Stack

It’s not needed that we need to use these tools however since we are already using some of the tool and we can think upon extending the tool for the max utilization.

PropertyGuru is already using some of the existing tools in the market for their continuous benefit however would like to propose some of the tools that we have come across which has much more benefit.

Here are the some of the tools that we would like to propose as part of the DevOps pipeline execution.

**GitLab**: We have proposed the Gitlab as well as part of the source control in the architecture. The GitLab has the end-to-end feature integrated right from the code commit, code deployment, code review, branching strategy applied, Review and approve the code, promotion of the code, Pipeline oriented execution, Repository container inbuilt to store the images.

**Datadog**: Since the data dog is already available, we can utilize the Datadog to monitor end to end applications. With the Metric Server monitoring, Instances monitoring, Application monitoring, Log Monitoring and all are available in a single dashboard. It’s also reducing to look on multiple dashboards for the monitoring purpose.

**AWS Managed Build tools:**  As an alternative if the GitLab is not opted proposing the AWS managed build tools for the Continuous Integration and Continuous Deployment.

**AWS Secret Manager:**

* AWS secret manage is a managed services and pay as you go.
* Since we are already using the AWS Databases and its tightly integrated
* AWS Secret manager allows key rotation which is manual in Hashicorp.
* Hashicrop is available as a open source and it does lay for the additional cost however we need the additional cost incurred in terms of scaling across
* Overhead cost for maintaining the Hashicorp Vault.

**AWS Proton**

A managed service for automated container and serverless deployment on the cloud.

## AWS Governance

**Authentication & Authorization**:

Authentication mechanism will be enabled for the validation of the user identity validation across with the domain users configured. IAM primary focus is with the access & the permission.

**Authorization**

Access rights for the individuals are configured for the controlling of the access across the resources /services in the AWS cloud. Not every user will have access to every application. The isolation of the access will be done through the Identity & Access management.

AWS Standard practices will be performed so that the project and the BU across the region are handled only by the respective projects.

**IAM:**

We can centrally manage users, security credentials such as access keys, and permissions that control which resources users can access. Using IAM

**Users** can access the assigned resources through the permission configured through the policies.

**Groups**: A group is being configured with the common group policies so that any policy applied are immediately traversed across all users in the group. The groups are customized for each project specific. This is customized based on the need of the requirement given.

Roles are configured for the complete isolation of the independent projects. Roles are not configured with any users however they can be intake or consumed by the authorized users. Different roles will be configured for the project specific execution.

**Policies:**

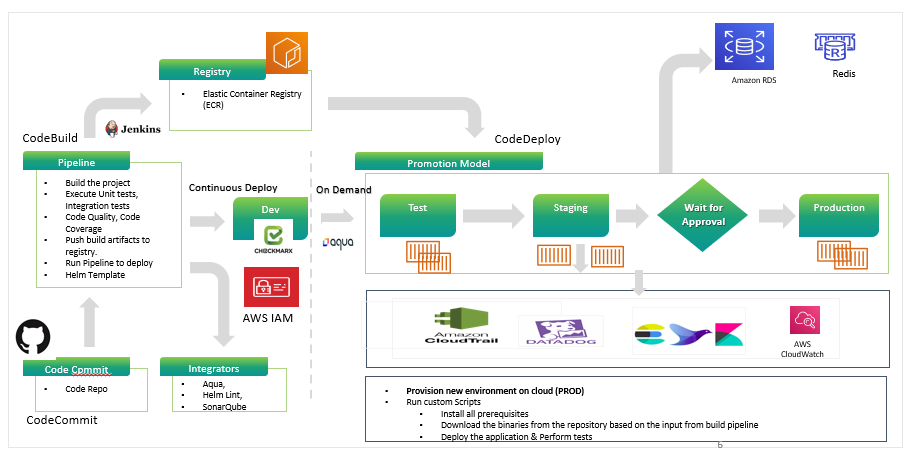
Policies are the one of the important key factor to maintain the roles and the permission across the cloud.

Policies are AWS objects attached to the users, groups, resources to define the permission that needs to be granted for accessing across the cloud.   
AWS policies in specific the following will be configured for the platform:

* Access Control
* Identity
* Resource level permission
* Service control policies
* Access control List
* Session Policies

Privilege Admin access will be configured so that the template layout, formation of the Cloud Formation Template of the services across the platform, validation of the users account and unique Id creation for the project, settings up the policies applied across the board.

## DevOps CI/CD Pipeline



The following are the steps executed as part of the pipeline:

The CICD pipeline is configured with all the required stages for the build execution.

The code repo will be configured to pull the code from the repository, and the code will be passed through the pipeline process for the execution.

Continuous Integration of the build process starts and triggers the staged build process.

The triggered build process on compilation is successful proceed with the next gated check with the code quality.

During the process the image is validated for all the security integration with the scanning using the tool. The image that is having the vulnerability criteria falls into the criteria like high, medium and low grouping.

The rules being set with the expectation for the code to pass in and also to upload on the Registry.

On successful compilation the build is uploaded onto the Container Registry.

The artifact is then called using the Continuous Integration process for the artifact deployment onto the K8 cluster.

Releases are done using the templates configured with the chart for the specific repos and the same can be reused across the board.

The releases are version controlled and the roll back can be done at any point of time.

All the artifacts are stored on the Container Repo and also, they are being tagged with the defined labels.

The promotion model is being set so that the same artifact can be promoted onto the next environment using the pipeline.

Test environment is setup as well so that the QA team can go and do their testing on the respective environment without disturbing any stage or production environment.

Performance test scripts are integrated to run on the code deployed and validated.

Alerts are being configured at all the levels in case of failure to trigger so that it’s completely monitored with the tools.

**Application and Infrastructure Monitoring**

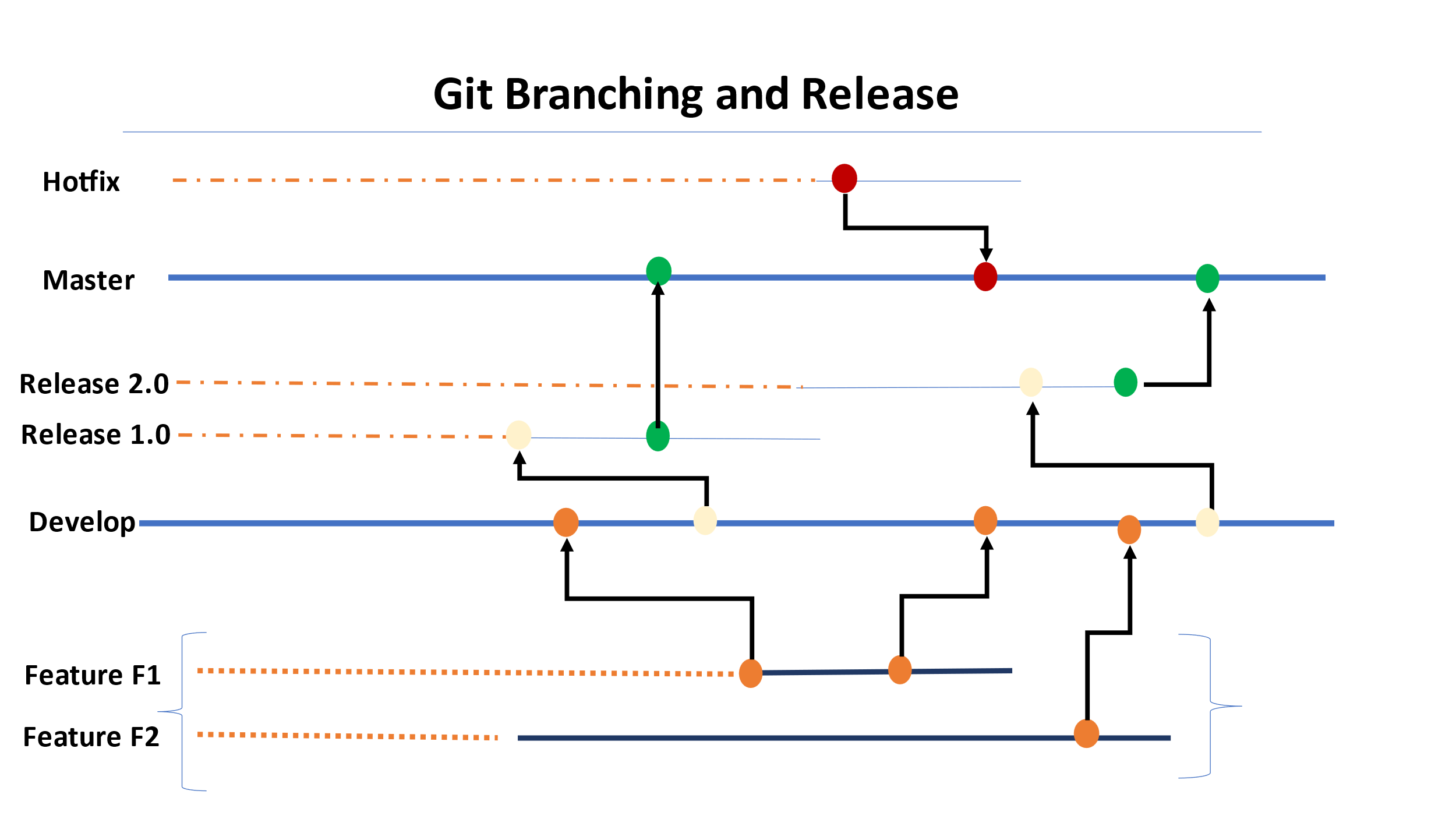
The application infrastructure monitoring is being done using the following tools integrated as part of the platform.

1. Datadog
2. CloudWatch

The Datadog tool will be tightly integrated with the Datadog with the agent configured for the entire monitoring. The agent installation on the monitoring will be retrieving all the data and the logs related to the application and infrastructure will be configured.

The APM (Application Performance Monitoring) will be integrated with the platform for the monitoring and all the services, and the pods will be integrated for the traces capture of the platform logs.

## Branching strategy



The most common strategies followed are the git flow and Trunk Based development.

* GitLab Flow
* Trunk Based Development

**Option 1:**

**GitLab Flow :**

The branching strategy consist of the following branches:

* Main Branches
  + Master
  + Develop
* Support Branches
  + Release Candidate branch
  + Hotfix branch

The main and the Develop branches are the most critical and continuous life cycle development phase. The other branches are in parallel development which are short lived branches.

The feature branches are shot lived branches which are being developed and merged onto the development branches.

The development branches are then being merged onto the supporting branches called as the Release branches.

Tags are being created for the branches to do the releases. There are standard naming conventions followed for the branching strategy so that the product is uniquely identified across the board.

The Release branches are the ones being merged onto the master branch. The master branch consist of all the changes from the feature releases to the candidate releases.

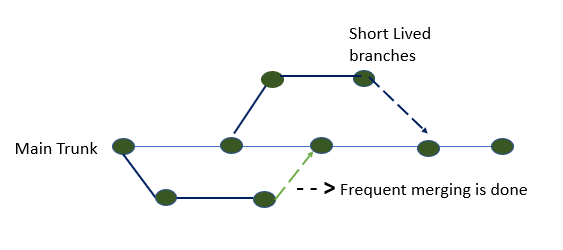
The releases are then Tagged based on the short outcomes of the releases.

The hotfix branches are created in case of any critical issues to be fixed on the production release being done from the master branch.

The master branch are then again branched out for the new changes in development/feature to worked upon by developers.

**Option 2:**

**Trunk Based Development**



Recently the changes are with the strategy aligned to get the development with the short-lived branches so that we have faster releases.

Trunk based development is a version control management practice where developers check-in their code to trunk or master branch.

The developers finish the new work on the feature, and they merge the changes on to the new code trunk.

There are challenges seen during this time with the conflict arising , Conflicts are increasingly complex as development teams grow and the code base scales. The trunk-based development model reduces these conflicts.

**Advantages**

* It allows continuous code integration - In the trunk-based development model, there is a repository with a steady stream of commits flowing into the master branch.
* It ensures continuous code review - Small commits of trunk-based development make code review a more efficient process. With small branches, developers can quickly see and review small changes.
* It enables consecutive production code releases - Developers should make frequent, daily merges to the master branch.
* it’s ready to deploy to production at any time.
* This gives team agility to frequently deploy to production. It also ensures that the further goals of production releases to be made much more frequent than the current process.
* Quicker to deploy and resolve the challenges rather than waiting for the long cycle completion.
* More collaboration during the execution and work as a one team with fully owned responsibilities.

**Recommendation**

**Option 2** is recommended as these are latest strategies followed across the organizations.

## Release Management (support for versioning)

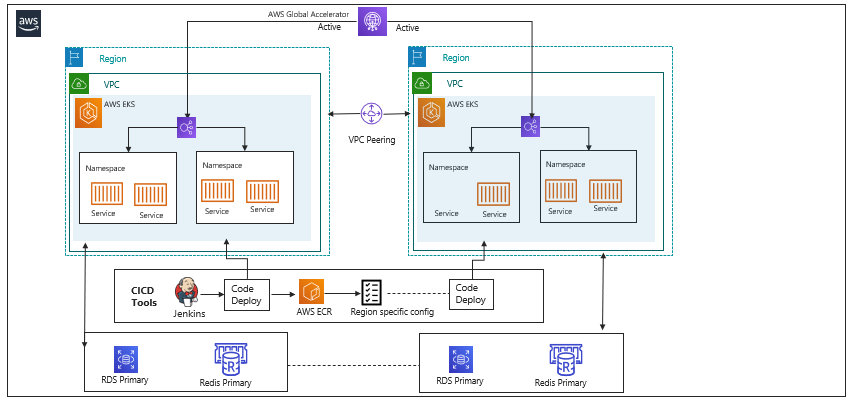
### Release Process

* Gates process will be created so that the respective team and the managers approval is obtained.
* Release processes are defined with the CICD Integrated
* Deployment will be done through the build tool.
* Frequent branches and the tagging process will be created for the deployment.
* Release will be tagged and stored as a Tag in the Repo.
* Hotfix branches will be created during the process of any critical fixes that needs to be part of the deployment.
* Communicate to the Stake holders on the release complete
* Helm chart will be used for the version control of the Releases.
* The Release will be deployed using the pipeline with the specific branching Tag created for the deployment.
* The Releases will be version controlled and also deployed from the specific pipeline.
* The promotion of artifact will be done using pipeline.

## DevOps Cloud Security for Application

* Security Virtual Private cloud will be configured.
* Security Rules for the private and public cloud configured.
* Access Identity and Management (IAM) Access Analyzer
* Key Management Service Integrated for encryption.
* S3 configuration and monitoring
* Security assessments on the firewall rules & configuration
* Access to the corresponding repos

## Disaster Recovery



The above diagram gives the solution running across the regions with the active mode of configuration done. The proposed model will ensure that all the required services are up and running.

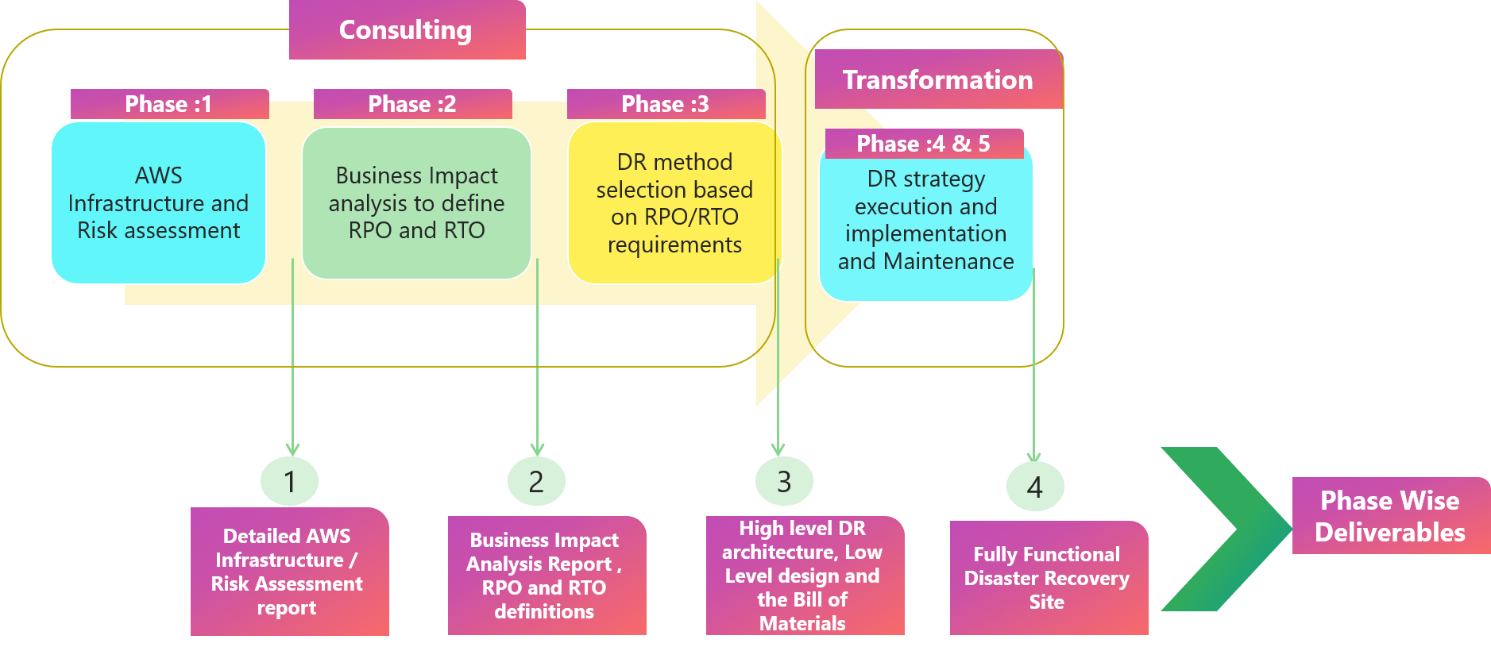
The backup and the pilot mode of execution will be integrated and executed during the process of execution.

The following are the flow will be enabled for the execution:

* Active- Active mode enabled for both the regions so that the recovery will be available always on with minimal or zero downtime.
* Real time execution is made available.
* Reduce the recovery time to the nearest zero down time.
* Backup frequency needs to be more due to the dependency on the recovery of current execution.
* The estimated backup will be depending on the RTO/RPO

The only disadvantage is that the cost of the AWS services is too high.

**Proposed Approach**



In this architecture, we will build the realize and the dependencies applications in AWS:

* AWS will be connected through AWS Direct Connect/VPN for secure access.
* Application data will be cloned onto the target regions and a continuous replication will be enabled.
* In this scenario, there are two parallel environments running for the platform.
* End user traffic will be routed using AWS managed services.
* Further, the responses sent from both the regions would be logged for analysis and validation.
* The platform will be tested in the AWS environment and will be cutover during a change window as per the scheduled.
* The DR process is being configured with the active – active mode where the clusters are being configured with the High availability.

**Continuous Integration and Continuous Deployment**

* Configuration code is configured in muti-region.
* Deployment of the infrastructure changes and configuration/code changes to be deployed at the same time with Primary Region.
* Analysis on the define RTO & RPO.
* Implement the DR method based on the previous results obtained.
* Code is stored in the repo will be trigged in the pipeline for the active application deployment .

Node Cluster Backup

* EKS cluster nodes are configured in the multi region for the DR process.
* The node cluster will be configured in two regions for the distribution.
* Distributed pods deployment in both the regions across the cluster
* Actively monitor the active regions are up and running.
* Template will be created for the deployment execution during the DR.

**Databases & Datastores**

* RDS distribution across regions will be configured during the initial setup.
* Database backups are configured regularly based on the discussion.
* The backup will be created using the AWS managed resources.
* DevOps will facilitate a Backup process for the regions.

**Other Options**

**Active - Passive setup** where the passive will be available in the warm setup so that we can switch over the Egress during the fail-over.

* Here only one region will be active at any point of time.
* The process remains the same as that of the above with the minimal changes.
* There is a minimal change done for the execution to make it Live for DR.
* Cutover will be performed with the analysis on the regions failover.

# Technology Stack – Indicative

|  |  |
| --- | --- |
| **Feature** | **Technology** |
| Programming language | Typescript and Java 11 |
| Cloud | AWS, CloudFront, EKS, ECS, Redis, IAM, RDS, MYSQL |
| Authentication | AWS Cognito |
| Networking | AWS Global Accelerator |
| API Gateway | AWS API Gateway |
| Microservices​ | NestJs for I/o bound functions and Java Springboot for CPU bound operations, Docker |
| Database/Storage​ | MYSQL, MONGODB |
| In memory Cache | Redis |
| CI/CD pipeline | Jenkins/GitHub /GitLab/AWS CodeCommit Pipeline, EKS, ECS, Kibana, EFK, DataDog, Prometheus |

# Schedule and Milestones

Following is the list of high-level schedule and tentative milestones. This is an indicative one and more detailed and accurate project plan shall be prepared during the discovery phase (please refer to the activities & deliverables of the Discovery phase in the Section-D document)

Project Start: T

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Tentative feature list** | **Schedule** |
| Milestone1 | * Listing search service * World API * Email template management service | T + 2 months |
| Milestone-2 | * Developer (Database Dependency) * Listing Management Service * User Engagement – Enquiry | T + 5 months |
| Milestone-3 | * Agent Subscription * User Service * Listing Management Service | T + 8 months |
| Milestone-4 | * User Service * Agent Management * Mobile BFF | T + 11 months |
| Milestone-5 | * Info Service * Agent Net | T + 13 months |
| Milestone-6 | * Advertising * Admin Service (Frontend) * Agent Net | T + 16 months |
| Milestone-7 | * Admin Service (Frontend) * My Activities | T + 18 months |

**Milestone deliverables**

Following are the high-level deliverables for the above-mentioned milestones.

* Source code and binaries
* SonarQube reports for the delivered services.
* Test cases, test reports, test coverage reports.
* Design document for the services
* Deployment scripts and config documentation

**Note:**

* Each milestone shall be followed by a UAT phase of 2 weeks where co-existence of monolith & new microservice shall be tested.
* Acceptance criteria for UAT is as mentioned in the below section.
* After successful completion of UAT, the new service shall be deployed in the production environment.

**Acceptance Criteria for above deliverables**

* Automated unit testing - 100% code coverage.
* User Acceptance testing – as per Acceptance criteria mentioned below.
  + Passing 100% of the user acceptance test cases with Zero Critical (P0) and High (P1) severity bugs identified
  + Defect Severity - Definition: Definition of Defect Severity and Priority are as below.

|  |  |
| --- | --- |
| **Defect Severity** | **Definition** |
| * P0 – Critical | * Defect may be a showstopper - that is, it stops the user from using the system further. |
| * P1 – High | * Defect occurs repeatedly and prevents the user from proceeding in the normal way, but a workaround exists. |
| * P2 – Medium | * A defect is isolated or does not stop the user from proceeding but is annoying and causing inconvenience. |
| * P3 – Low | * A defect that in no way affects the performance or functionality. E.g.: Aesthetic issues and grammatical errors in messages. |