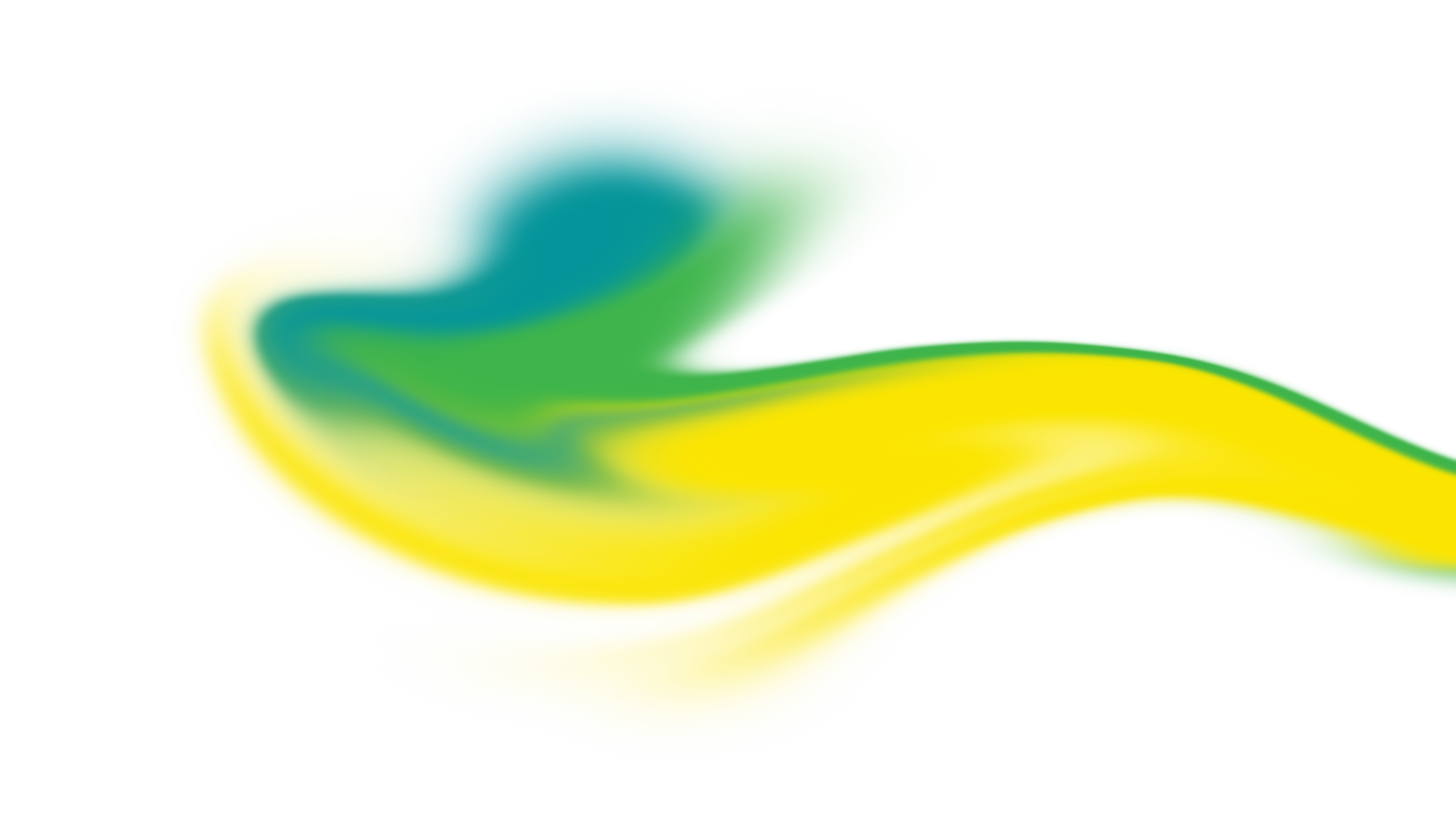


**CI/CD DevOps pipeline automation**



**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 Freight Verify 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 Freight Verify that it may have access to during this interaction.

**Disclaimer**

This document has been prepared based on the information provided by Freight Verify. 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.

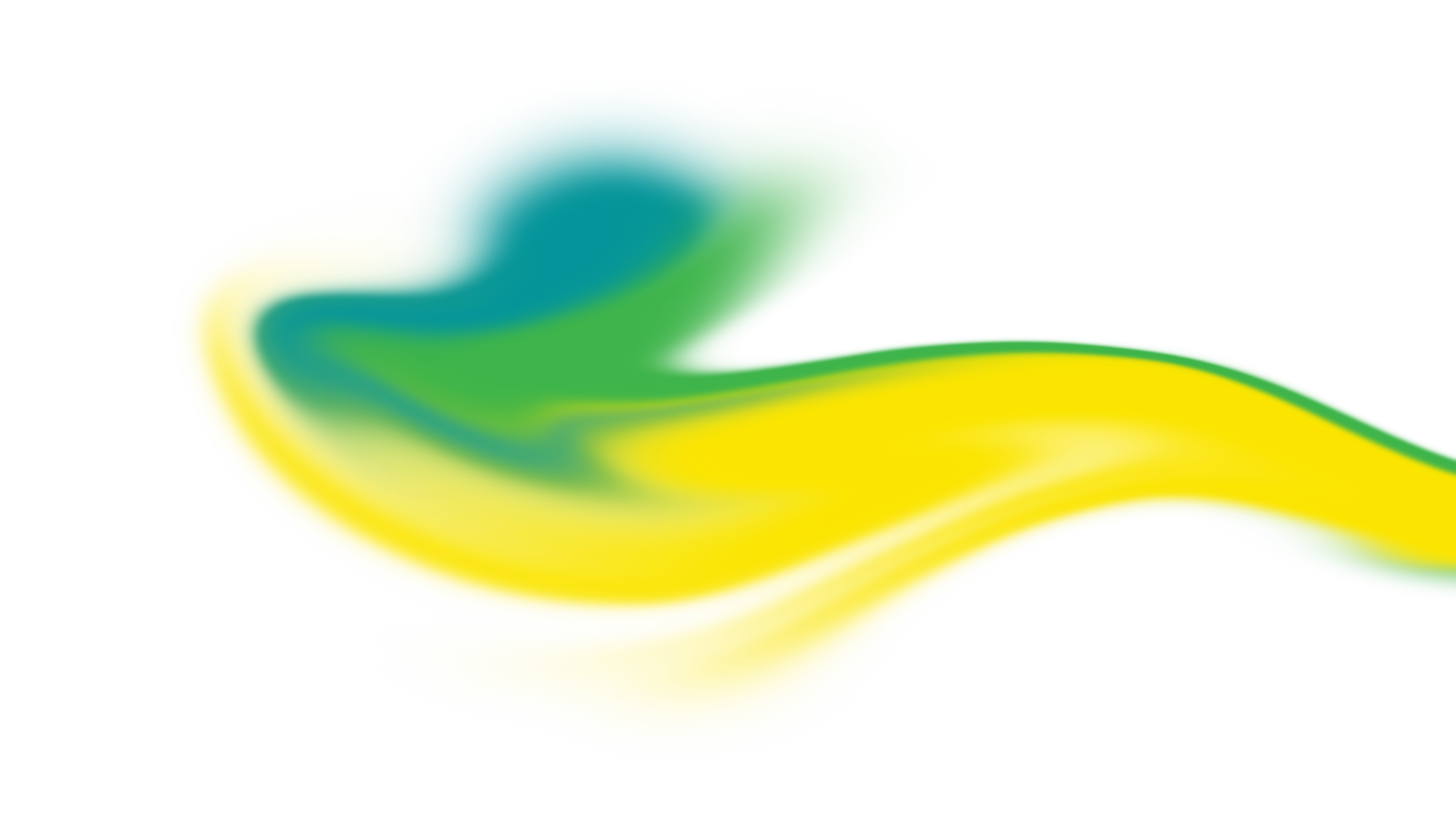


Table of Contents

[1. Executive Summary 3](#_Toc902678126)

[1.1. Background 4](#_Toc1588470103)

[1.2. Our Understanding of Requirements 4](#_Toc2121725283)

[1.3. Functional Requirements 6](#_Toc1469272287)

[1.4. In Scope 6](#_Toc537737901)

[1.5. Out of Scope 7](#_Toc1110989407)

[2. Proposed Solution 7](#_Toc1333839471)

[2.1. Tools / Technology 8](#_Toc1263377736)

[2.2. Solution Architecture 9](#_Toc1230379800)

[2.3. Infrastructure As Code 10](#_Toc1576007676)

[3. Deployment Strategy 10](#_Toc1282349788)

[3.1. Blue- Green deployments 11](#_Toc26322477)

[3.2. Canary deployments 12](#_Toc969981240)

[3.3. Quality Gatekeeping 12](#_Toc361509536)

[3.4. Assumptions 13](#_Toc1906623455)

[3.5. Dependencies 14](#_Toc254565643)

[3.6. Testing & Validation 14](#_Toc1093330363)

[3.7. Risk and Mitigation Plan 15](#_Toc1317672816)

[3.8. Execution Schedule 16](#_Toc1434214156)

[3.9. Project Communication Model 16](#_Toc1901449155)

[3.10. Change Request Management 17](#_Toc1150599831)

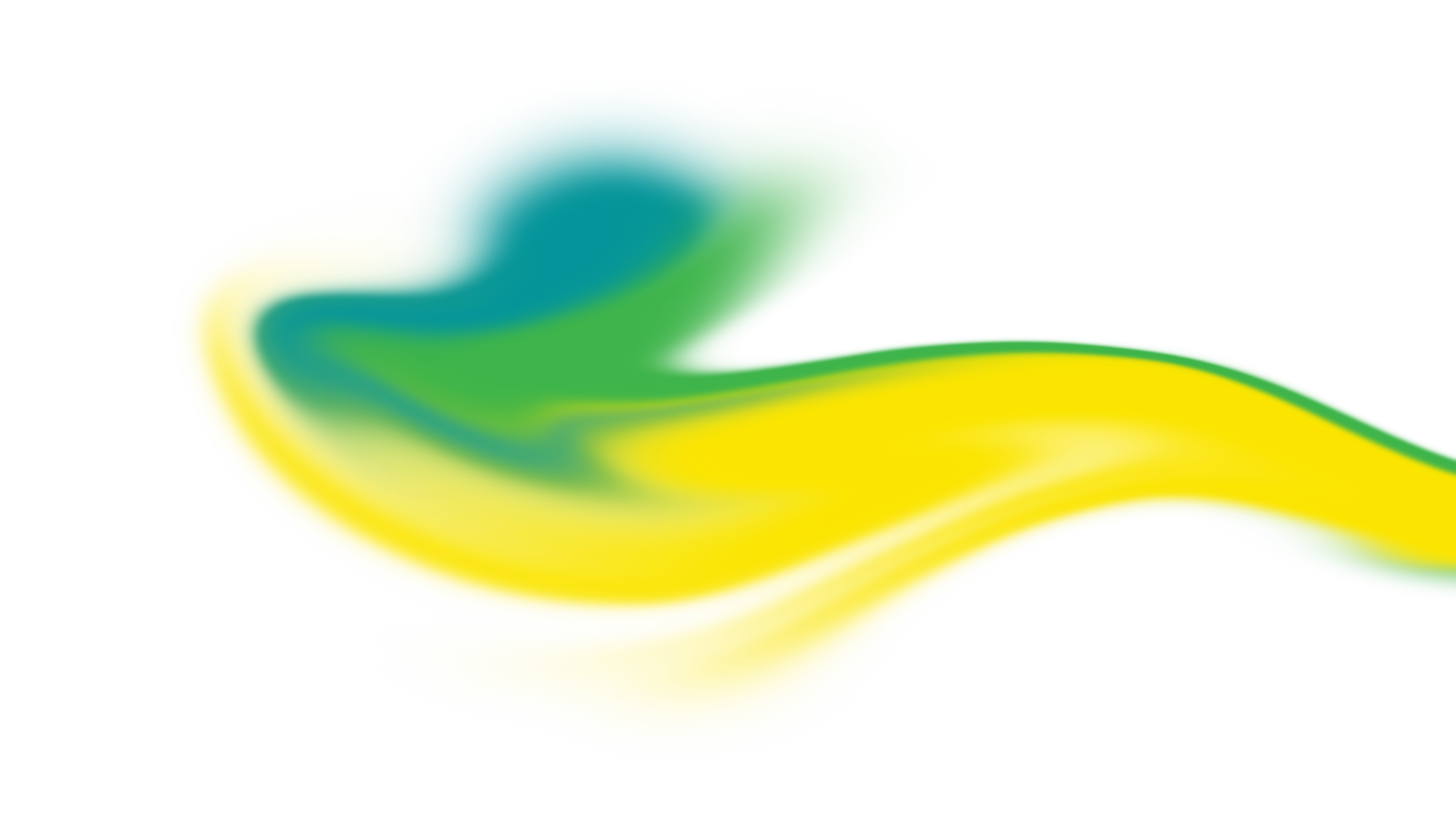
[3.11. Acceptance Criteria 17](#_Toc1168666497)

[4. Commercials 17](#_Toc1764714303)

[4.1. Cost for Development 18](#_Toc169571111)

[4.2. Terms and Conditions 18](#_Toc201093653)

[5. About Happiest Minds Technologies limited 18](#_Toc1760377118)



# Executive Summary

Happiest minds would like to thank FREIGHT VERIFY for giving us this opportunity to bid for this RFP for the CI/CD DevOps automation pipeline. FREIGHT VERIFY aims to implement a comprehensive Continuous Integration and Continuous Delivery (CI/CD) solution utilizing DevOps practices on their existing infrastructure. The primary objectives of this implementation are to:

* Increase deployment velocity and software delivery speed
* Improve reliability and reduce errors in software releases
* Implement infrastructure-as-code for consistency and efficiency
* Leverage automation, monitoring, and security at all stages
* Support modern application architectures using containers and orchestration

Freight Verify aims to establish a robust CI / CD pipeline powered with Build automation, monitoring, and latest deployment strategies and ensuring all the industry best practices are incorporated for their development lifecycle. Drawing from our extensive track record of successful collaborations with numerous clients in a comparable industry, we are confident that Happiest Minds can offer a highly effective solution to bridge existing gaps and propel the business towards substantial growth. We eagerly welcome the opportunity to submit a proposal in response to Freight Verify’s RFP, firmly believing that Happiest Minds is the ideal choice as your strategic partner.

## Background

Freight Verify has existing system and platform that needs to be integrated with a newly proposed DevOps approach utilizing continuous integration and continuous deployment (CI/CD). The goal is to implement automated testing, infrastructure as code, and deployment automation to improve software quality and delivery velocity. The proposal outlines an approach to implement CI/CD and DevOps best practices leveraging various tools and technologies to enable automated, reliable and secure application builds and deployments for the Freight Verify existing environment. The scope focuses on pipeline automation, infrastructure-as-code, release strategies and monitoring/alerting while excluding major migrations or rewrites.

## Our Understanding of Requirements

Our understanding of the requirements is based on the initial conversations with FREIGHT VERIFY key members and based on the artifacts shared for reference. The overall outcome is to build a robust CI/CD pipeline using Jenkins across the entire stack - from source code to production. Key highlights as part of this implementation will be the following.

* GitHub/GitLab for source code management and Jenkins for orchestrating builds and releases.
* Automated unit, integration and performance testing in the pipeline to improve code quality.
* Containerizing applications using Docker & managing containers with K8S (EKS) for portability & scalability.
* Managing underlying infrastructure as code through Terraform for efficiency and consistency.
* Improved monitoring, logging and alerting using tools like CloudWatch.
* Implementing Blue/Green and Canary deployments to reduce downtime and risk.
* Security and compliance checks baked into the pipeline.

The expected outcomes are faster and more reliable releases, quicker recovery from failures, and improved visibility and control over the software delivery process.

Compliance checks be integrated into the pipeline using InSpec??

Below are the project goals, expected project outcomes & best-in-class approach to realize the desired objectives

|  |  |  |
| --- | --- | --- |
| Project Goals | Project outcome | Approach to Achieve |
| Continuous Integration and Continuous Delivery Pipeline | Implement a CI/CD pipeline for automated building, testing and deployment of applications from source code to production environments. | * Utilize Jenkins as the primary workflow orchestration engine to coordinate various stages in the pipeline. * Integrate Jenkins with source code repositories (GitLab) for tracking changes and triggering automated workflows. * Incorporate build tools like Maven and runtime platforms like Java/NodeJS for application containerization. * Enable automated unit, integration and performance testing in the pipeline to validate code quality. * Leverage Docker for containerizing applications and Kubernetes for container orchestration. * Implement automated infrastructure provisioning and application deployments. * Facilitate rollbacks and ensure immutability of infrastructure and artifacts as they progress through environments. |
| Infrastructure as Code (IAC) | Manage underlying infrastructure for efficiency & consistency | * Manage infrastructure through Terraform templates * Maintain environment parity between dev, test, stage and production for standardization. * Version control and peer review infrastructure as code definitions. * Automated infrastructure provisioning in pipeline stages. |
| Alerts and Monitoring capability |  | * Incorporate logging, monitoring and alerting capabilities using tools like CloudWatch, or ELK stack. * Monitor application KPIs, infrastructure health, audit trails and logs. * Setup thresholds & alarms for critical incidents and notify users. * Provide dashboards for operational visibility of systems and workflows. |
| Release management & Deployment strategies |  | * Implement Blue/Green & Canary deployments to reduce downtime. * Facilitate incremental rollout to user subsets for early feedback. * Incorporate automated rollbacks and leveraging of feature flags. * Seamless coordination across infrastructure, application, database and messaging tiers. |

## **Functional Requirements**

* Configure CI/CD pipeline for build automation, automated testing, artifact management and deployments
* Implement infrastructure-as-code with Terraform to provision AWS resources
* Utilize Docker and Kubernetes (EKS) for containerization and container orchestration
* Leverage AWS services like EC2, ALB, ECR, S3, CloudWatch, etc.
* Integrate security controls and compliance into pipeline
* Implement blue/green and canary deployment strategies
* Instrument monitoring, logging and alerting using CloudWatch

## In Scope

While the scope of this proposal focuses on implementing a future-ready CI/CD pipeline and DevOps culture, major application rewrites or migrations are excluded. With this proposal, Freight Verify aims to gain enhanced automation, monitoring, and deployment best practices for their development lifecycle.

* Set up CI/CD pipeline for build, test, store, deploy automations
* Containerize applications using Docker and deploy on EKS
* Manage infrastructure and AWS resources as code with Terraform
* Implement blue/green and canary release strategies
* Integrate security and compliance checks in pipeline
* Monitor deployments and integrate alerting
* Deployment Pipeline Testing & Validation

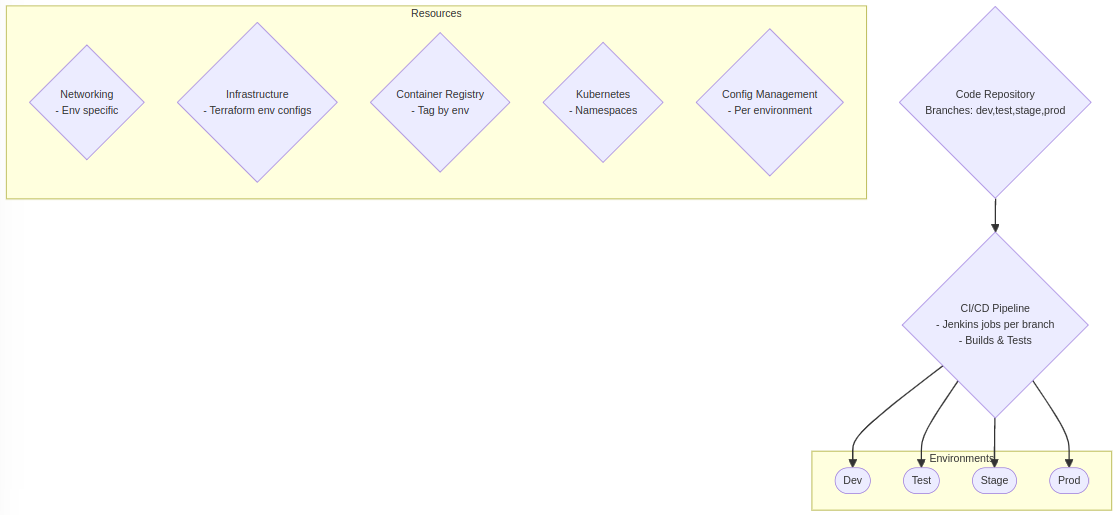
## Out of Scope

* Any deviations from the mentioned features / requirements above.
* Security scanning, IaC scans, artifact scans, Container scanning at various stages.
* Pipeline Compliance evaluation & maturity assessments.
* Capture and audit environment-specific deployment logs
* Check for regulatory compliance PCI DSS, HIPAA based on the application.
* Changes to the underlying structure and architecture of the existing applications
* Dynamic Application Security Testing (DAST) & SAST to scan application binaries & detect vulnerabilities.
* Overall IT security responsibilities (to be handled by FREIGHT VERIFY IT teams)
* User Manual, SOP, Runbook
* Any tool/ third party integration other than mentioned on the requirement section

# Proposed Solution

**CI/CD pipeline**

By leveraging the best practices around Environment management, Configuration management, parameterization, controls around infrastructure/code/artifacts per environment, Blue Green deployment strategies and end-to-end automation, we can model the distinct environments securely within a single CI/CD pipeline.



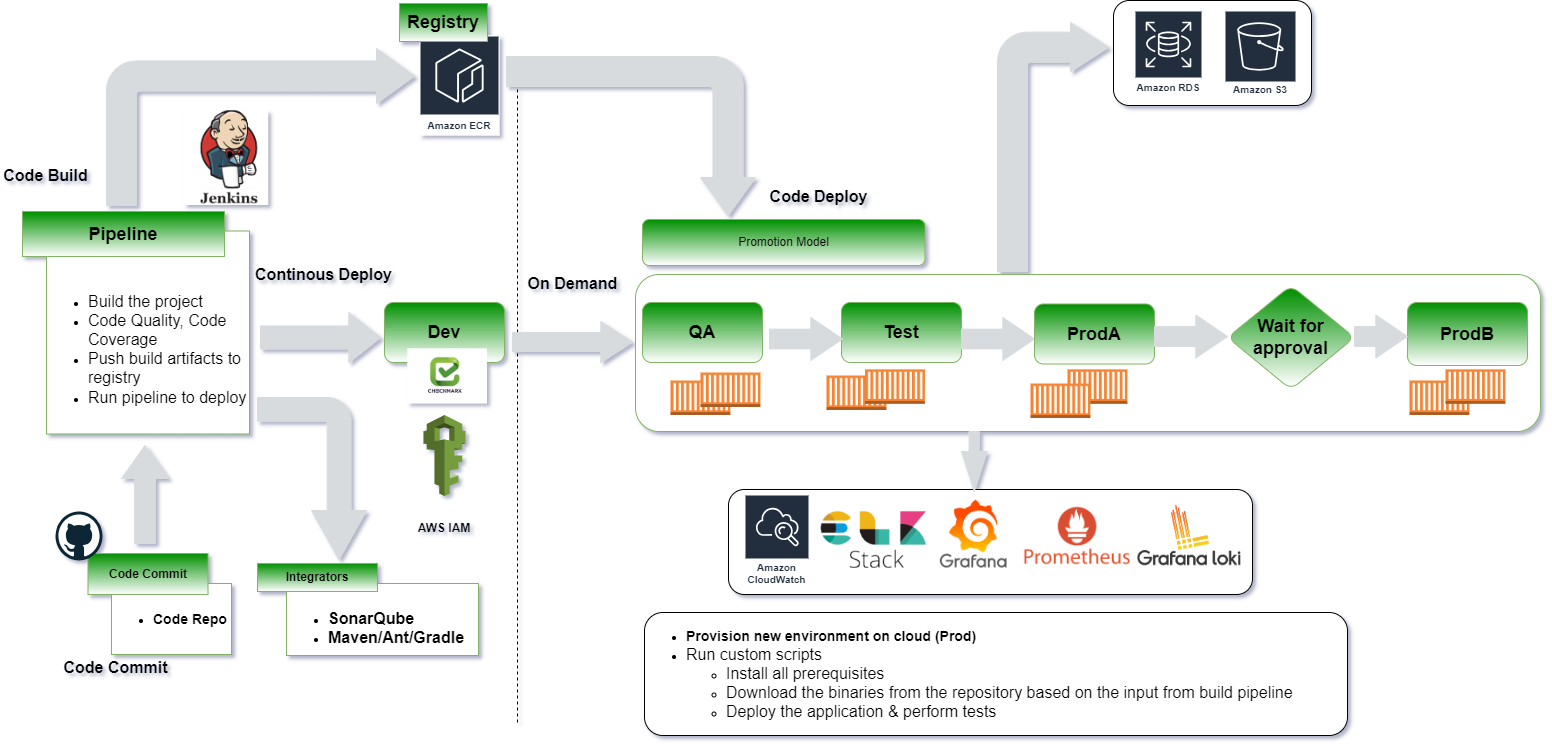
|  |  |
| --- | --- |
| **Area of interest** | **Best practices** |
| Source code management | * Use separate branches in version control for each env (dev, test, stage, prod) * Implement pull requests and peer reviews for code promotion across branches * Automate branch creation and merging where possible |
| Testing | * Integrate automated unit, integration, and UI testing in CI pipeline * Run regression testing across environments before production deployment. |
| Infrastructure Provisioning | * Manage infrastructure as code with separate config per environment * Provision parallel isolated environments across regions/VPCs/subnets * Leverage Docker tagging and container registries to push images per environment. * Switch Staging & Production environments after releases for efficiency * Limit access and implement least privilege controls for staging/prod. |
| Configuration management | * Parameterize Jenkins jobs, Terraform templates, Kubernetes manifests for reusability. * Configure Jenkins jobs per branch for environment specific builds & deployments. * Maintain environment parity but allow configurable differences * Use Kubernetes namespaces to isolate resources between environments. * Implement central config store or management tools |
| Artifacts & Binaries | * Create environment-specific build artifacts and container images * Tag container images and version binaries for traceability * Use a dedicated container registry per environment |
| Deployment Strategies | * Implement blue-green, canary deployment strategies. * Maintain environment parity as much as possible for release confidence. * Implement zero-downtime deployments per environment * Follow a dev -> test -> stage -> prod deployment workflow, with approvals. * Perform zero-downtime deployments where possible * Seamless shift of traffic from old to new version |

## Tools / Technology

|  |  |  |
| --- | --- | --- |
| **Tool / technology** | **Purpose** | **Commercial / Open source** |
| Gitlab | GitLab for managing source code, version control and collaboration | Open source |
| Maven | Maven for building the Java-based microservices & React JS applications. |  |
| Docker | Docker will be used to containerize the microservices. |  |
| Terraform | Infrastructure updates will be handled using Terraform, triggered by the Jenkins CI/CD pipeline. | Open source |
| Jenkins | Continuous Integration and Deployment: Jenkins will be used to establish a robust CI/CD pipeline. It will integrate with the version control system (e.g., Git) to trigger automated builds, perform code quality checks, and deploy the applications to the EKS cluster. Jenkins pipelines will be defined using declarative syntax for easy maintenance and scalability. | Open source |
| Prometheus |  | Open source |
| Grafana |  | Commercial |
| SonarQube | SonarQube (formerly Sonar) is an open-source platform for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs. | Open source |
| Slack |  | Commercial |
| AWS Elastic Container Registry (ECR) | AWS Elastic Container Registry (ECR) will be used to store and manage Docker images. | Commercial |

## Solution Architecture

Below is the proposed solution architecture / technology landscape to achieve DevOps automation.



## Infrastructure As Code

This approach provides portability of applications via containers/Kubernetes while also keeping the underlying infrastructure consistent and compliant through IaC.

**Containerization with Docker**

* Containerize each microservice, backend API, and frontend as needed for modular deployment.
* Leverage Docker files to define OS, dependencies, configs needed for each container.
* Build images through automated Docker builds using Docker hub/registry.
* Follow strategies like multi-stage builds and small container images to optimize images.
* Store application config, secrets external to containers and inject through K8s config maps /secrets.
* Instrument containers for health checks, metrics, tracing.

**Kubernetes Deployments & Management**

* Provision Kubernetes cluster (EKS) with required capacity and availability.
* Define Kubernetes yaml files for pods, services, deployments, ingresses etc.
* Set up CI/CD pipeline to build images and push to registry.
* Deploy containers to Kubernetes clusters from registry using Kubernetes manifests.
* Implement Helm charts to package and deploy microservices.
* Use namespaces and network policies for isolation. Auto-scale pods based on usage.
* Centralized logging, monitoring with tools like Prometheus, Grafana.

**Infrastructure as Code with Terraform**

* Codify infrastructure requirements and configuration as Terraform templates.
* Create reusable Terraform modules for each environment and resource.
* Manage Terraform state in remote backend for consistency.
* Provision core resources just-in-time like VPCs, subnets, security groups.
* Automate infrastructure changes through CI/CD pipeline invoking Terraform.

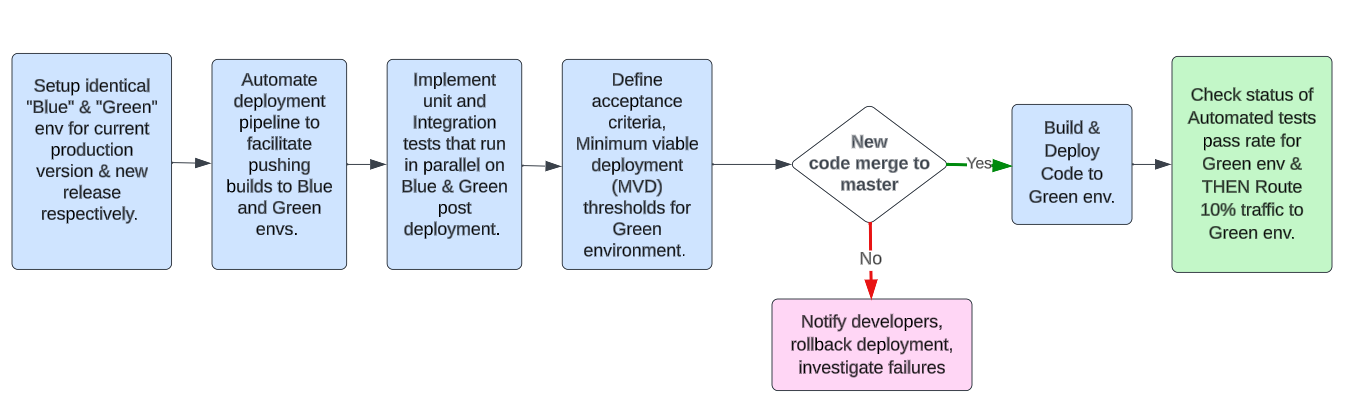
# Deployment Strategy

Blue-green deployment is a release technique that reduces downtime and risk by running two identical production environments called Blue and Green. At any time, only one of the environments is live, with live traffic redirected between them. Canary deployment incrementally shifts traffic to a new version by first rolling it out to a small subset of users. Blue green enables rapid launch and rollback minimizing downtime, while canary catches potential problems early through incremental rollout monitoring.

Implementing a combination of blue-green and canary deployment strategies provides a robust deployment strategy and complementary benefits that together enable safe, controlled, and smooth software releases. Blue-green deployment facilitates rapid rollout of new versions while eliminating downtime by shifting traffic atomically from the old version to the new. This allows easy rollback if issues arise.

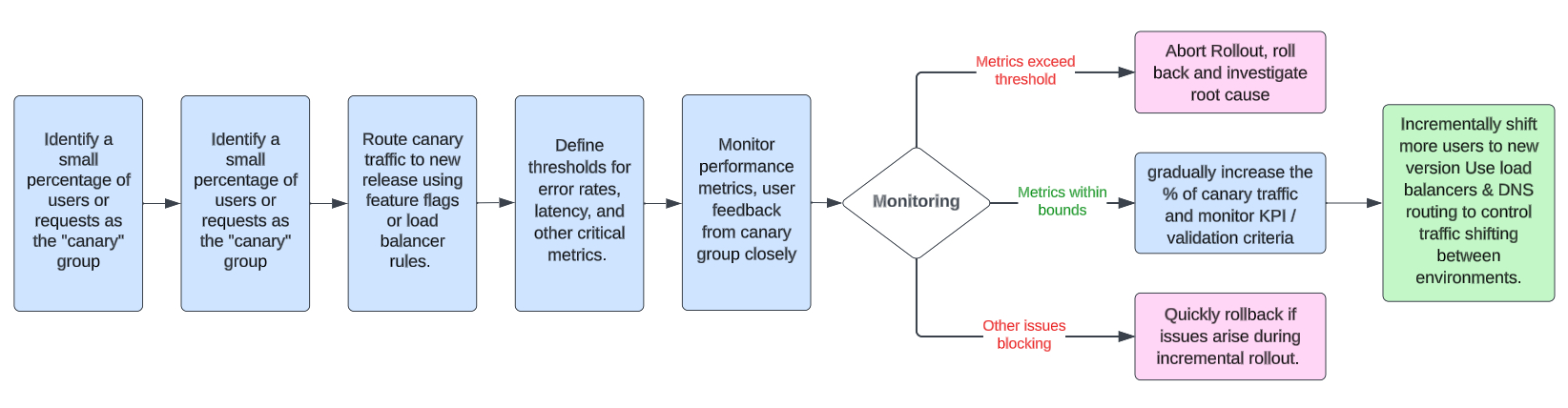
Canary deployment incrementally shifts a small percentage of traffic to the new version, allowing time to detect flaws or regressions through close monitoring before exposing all users. While blue green enables quick time-to-market and rollback, canary offers early warning indicators and fine-grained control for more cautious releases. Using them in conjunction allows us to validate new versions with a subset of users through canary analysis while also achieving rapid switching for the broader user base via blue green. This defense-in-depth approach reduces risk, minimizes disruption, and creates a robust, resilient release process aligned with our governance policies. The blend of techniques provides optimal stability, visibility, and flexibility.

## Blue- Green deployments

With a Blue-Green deployment model, we will maintain two identical environments: the Blue environment represents the current stable version, while the Green environment represents the new version or feature being deployed. The automated deployment pipeline will be built with Jenkins to facilitate the deployment process. Then the automated testing suite triggers will be integrated with the pipeline that runs against both Blue & Green environments to ensure functional and performance validation. Based on the release criteria definition and the established rollback strategy it's possible to quickly revert to the stable version (Blue). In case of any issues or unexpected behavior during the deployment, the rollback procedure takes control of the workflow. The traffic will be gradually switched from Blue to Green environment using load balancers or DNS routing. This helps us in minimizing potential downtime and allows real-time monitoring and validation. By continuously monitoring key performance indicators (KPIs) and tracking user feedback, the success rate of a deployment is increased and helps in identifying any potential issues that may require further attention / debugging. 

## Canary deployments

In addition to Blue-Green deployments, a Canary deployment strategy will be incorporated to gradually roll out changes to a subset of users or systems before a full release. This will allow for early detection of any issues or performance problems. The strategy includes identification of a subset of users or systems that will act as the Canary group for each release. The feature flagging or traffic routing mechanisms to direct a portion of traffic to the Canary group is incorporated while the majority of users remain on the stable version. Key metrics and performance indicators are monitored for the Canary group, comparing them against predefined thresholds and service level objectives (SLOs). Once the metrics and KPIs are close to the expectations, the proportion of traffic served to the Canary group is gradually increased (ex. 10%, 25%, 50% etc.). As the release progresses, closely monitoring the impact on metrics and user experience becomes critical. Based on performance and feedback received, owners decide whether to proceed with a full release or roll back the changes.



Pre-deployment testing on the staging environment is required for both strategies before releasing to production. The Blue green strategy relies on automated or manual smoke testing after shifting full traffic to the target environment. Canary deployment strategy requires more sophisticated validation checking how new version impacts a subset of users. By combining both strategies, the best of both worlds help in identifying issues early and also help in minimizing downtime for broader releases. This results in reduced risk, smooth user experience, and high quality of deployments.

## Quality Gatekeeping

SonarQube will be integrated into the CI/CD pipeline to perform static analysis on the source code and provide code quality metrics. SonarQube scans will run during the build stage and analyze the codebase for bugs, vulnerabilities, code smells, and other issues. The scan will evaluate code coverage provided by unit and integration tests. Rules in SonarQube quality profiles will be configured to match project standards. Any new code changes merged into the mainline branch will trigger a SonarQube scan as part of the CI pipeline. The scan results will be recorded on the SonarQube server and made available via dashboards.

SonarQube quality gates will be defined with thresholds for metrics like unit test coverage, duplication percentages, and security ratings. If the scan results violate any quality gate criteria, the build will be failed by SonarQube. This forces developers to fix issues before their code can be merged and progress further down the deployment pipeline. Quality gates ensure that only clean, quality code that meets the standards gets deployed. All source Code will have unit test coverage per company guidelines. Unit tests will run during the “Build” stage of the pipeline. Integration tests validating end-to-end functionality will run after unit testing is completed.

|  |  |
| --- | --- |
| **Application Monitoring KPIs** | **Infrastructure Metrics / KPIs** |
| * Request throughput (requests/sec) * Error rates (4xx/5xx responses) * Response times (latency) * Traffic volume (unique visitors, page views) * Conversion rates (e.g., signups, purchases) * Payment transaction success rates * API uptime and availability * End-user response times * Application exception rates | * CPU utilization * Memory usage * Disk I/O rates * Network throughput * Database connections/latency * Cache hit rate * Node/container health checks * Auto-scaling events * Infrastructure uptime/availability |

**Note**: Above are some of the potential metrics / KPIs that can be considered. This will be researched in depth during the brainstorming process with Freight Verify.

In addition, Test coverage metrics will be collected and reported in SonarQube as part of the static analysis. Test execution and coverage is essential to analyze code risk, gaps, and validate readiness for downstream environments. By integrating SonarQube analysis into the CI/CD pipeline along with comprehensive test coverage, code quality and security can act as automated gates to promote or fail deployments even before reaching staging or production. This catches issues early and ensures standards are met.

## Assumptions

* Provision of automated unit tests / framework along with integration test will be Freight Verify’s ownership
* Only FREIGHT VERIFY authorized users can access and download the reports for security and to control access.
* FREIGHT VERIFY will address all the dependencies as per the agreed timelines during the start of the project.​
* FREIGHT VERIFY should provide sign off for requirements to be implemented by first week of project start.
* The Blue-green environments provided by Freight Verify should be identical to avoid configuration drift. Requirements like operating systems, dependencies, configurations should match.
* Existing infrastructure with AWS lambda functions & Kafka implementation will remain as-is untouched.
* Response to Happiest Minds’ queries within two business days.
* The requirements will be based on the points discussed during RFP along with the business expectations. Detailed evaluation will be done immediately after the project kickoff meeting.
* Scope and Effort will be revisited for additional details or deviations to existing ‘in scope’ section may have an impact on timelines and costs for any deviations to scope or requirements.
* Happiest Minds personnel will follow Happiest Minds holiday calendar of the respective base location. ​

## Dependencies

|  |  |
| --- | --- |
| **Dependency Factors** | **To be available by** |
| Product owner for the knowledge transition of the application, clarification, and review of the user stories.​ | Start of the project |
| Designate a single point of contact (SPOC) available for Happiest Minds team throughout the course of the engagement to facilitate technical discussions with all client stakeholders for business and technical requirement discussions and to resolve any dependencies or questions raised by the team | As needed during the project |
| FREIGHT VERIFY to review and sign off all detailed requirements within the first week of project implementation. | Start of the project |
| Provide any coding guideline or standards that happiest minds must follow for component implementation along with Code repository access | Start of the project |
| Test strategy /plan/ cases shall be reviewed and accepted by FREIGHT VERIFY before test execution. | As needed during the project |
| Review UAT Test Case document | As needed during the project |
| Response to Happiest Minds’ queries within two business days. Any delays in response might have an impact on the effort and schedule & will be treated as a Change Request, | As needed during the project |
| Feedback on the deliverables for each milestone must be provided within 3 days of the delivery by Happiest Minds, else, they shall be deemed accepted | During entire project engagement |
| AWS Account & instance for development, testing and production environment setup and deployment related activities, DEV and Test environments | Start of the project |
| The review comments by FREIGHT VERIFY stakeholders on design / proposals will be provided. | Within 48 hrs. |

## Testing & Validation

The test strategy is to primarily validate the deployment strategies to ensure seamless transition to the new version releases and incrementally validate deployment through continuous testing while also catching issues as early as possible via progressive rollout monitoring. This balances speed with safety for releases.

Testing will be automated as much as possible for consistency and efficiency. Automated unit / UI / API tests will supplement integration tests for system testing. Test data will be generated for repeatable tests across environments. Continuous Validation strategy will be in play with Lower environment deployment triggering automated test suites to provide continuous validation. Tests will also run after production deployment as a final check. Monitoring in Production will be ongoing as part of Continuous testing in addition to pre-deployment testing.

* **Unit Tests** - Unit tests will be written by developers for every new feature/component to validate logic and prevent regressions. These will run during code build.
* **Integration Tests** - API/service integrations will be validated via automated integration tests after unit testing. Critical user journeys and workflows will be tested.
* **System/E2E Tests** - End-to-end testing will simulate production-like behaviour across the entire system including infrastructure. Automated browser tests will be used.
* **Performance Tests** - New features will undergo performance testing under production-scale load and data volumes in the staging environment.
* **User Acceptance Testing** - Hands-on UAT will be conducted in the staging environment for business user validation before production deployment.

Production monitoring will be implemented using tools like Prometheus and Grafana dashboards to measure performance and serve requests. This helps catch any issues not detected during earlier testing. By combining continuous automated testing, UAT, and production monitoring, the solution provides complete validation of changes across the entire lifecycle to prevent defects and ensure high quality. In addition, testing shall also focus on validation of the deployment strategy to ensure high Quality standards. Below is the test approach for the deployment strategy.

**Blue-Green Testing**

* Setup mirrored "blue" and "green" environments for adequate testing.
* Perform end-to-end automated tests on green environment after code deployment.
* Execute performance, load and staging tests on the green environment.
* Conduct integration testing across all tiers of the application in green.
* Run automated smoke/sanity tests on green after cutting over traffic.
* Monitor application KPIs and infrastructure after full rollout to validate deployment.
* Route a small amount of real production traffic to green environment for user acceptance testing.

**Canary Testing**

* Run canary builds through the same automated test suites as main builds.
* Deploy canary build to production alongside current build.
* Use feature flags to direct small percentage of traffic to canary.
* Monitor performance KPIs and business metrics from canary group closely.
* Compare canary metrics against baseline metrics for conformance.
* Test canary build iteratively with increasing percentage of traffic.
* Abort rollout if metrics deviate outside expected thresholds at any point.
* Gather direct user feedback from the canary group through surveys or interviews.

## Risk and Mitigation Plan

* Any risks identified during project execution would be discussed with FREIGHT VERIFY SPOC immediately
* From risk identification through risk response planning and status update during risk monitoring and control.
* These risks would be highlighted in the weekly status report as well, along with other open items/issues.

## Execution Schedule

The delivery schedule is based on our current understanding of the features and requirements. Any significant changes in scope of the project are expected to have an impact on the cost and schedule of the project.

The proposed delivery schedule for this project is presented below. The project is estimated to be executed in a span of XX weeks. Following are the recommended deliverables with the schedule and the outcomes.

**Assume Project Start Date: (T)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Due Date in Weeks** | **Owner** | **Acceptance required from the client** |
| Discovery phase | T+ | Happiest Minds | Yes |
| FREIGHT VERIFY CI / CD pipeline building and end-to-end testing and deployment | T+ | Happiest Minds | No |
| User acceptance testing | T+ | Happiest Minds | Yes |

## Project Communication Model.

**Project Communication Model**

The communication model will involve the activities below for successful delivery of the project:

1. **Daily Stand-up meetings**: Between Offshore team and Project manager to discuss tasks completed, task planned and any outstanding blocking issues. FREIGHT VERIFY is not required to participate in these meetings, but they are welcome to participate if they want to.
2. **Weekly Status Meeting**: Between Project manager at Happiest Minds and Project manager of FREIGHT VERIFY to discuss weekly status report and next week’s plan, issues, risk on the on-going phase of the project.
3. **Monthly Operational Meeting**: Between Delivery Manager and Project Manager at Happiest Minds and Project sponsor at FREIGHT VERIFY to discuss progress on the project, issues, and risk with resolution for project delivery and commercials. Monthly meetings can be planned if there are plans to continue further enhancements.

## Change Request Management

Changes to the scope will mean any of the following:

* Any changes to the scope of the project as detailed in section “In Scope”
* Invalidation of any of the assumptions detailed in section “Assumptions”
* Any change to the terms and conditions as defined in Commercials and Payment Terms sections
* Non-fulfillment of any of the dependencies detailed in the sections Dependencies

In case of change request, the scheduled end date for this Project and/or the fees associated may change. Whenever a change is identified, it will be managed as per the below process:

* For any changes to the scope, either FREIGHT VERIFY or Happiest Minds will submit a Change Request
* Happiest Minds will issue a Change Order providing the impact of the change to the schedule and/or fees
* FREIGHT VERIFY SPOC will review and either approve or cancel the change order
* Changes will be implemented only after FREIGHT VERIFY SPOC approve & sign of change order form.
* For any FREIGHT VERIFY dependencies that are not met or issues that are not resolved, which could impact the schedule – Happiest Minds Project Manager will complete a Change Order and inform the FREIGHT VERIFY SPOC.

## Acceptance Criteria

* The UAT (User Acceptance Testing) document would be shared to FREIGHT VERIFY and signed off 2 weeks before the start of Acceptance Testing
* Acceptance testing shall be performed mutually by Happiest Minds & FREIGHT VERIFY.
* The acceptance criteria will be successful if there are 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 | Defects occur repeatedly and prevent 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. |

# Commercials

## Cost for Development

|  |  |
| --- | --- |
| **Description** | **Amount in INR** |
| Cost for   * Discovery & Implementation * Warranty & KT |  |
| **Total** |  |

**Payment Schedule**

Happiest Minds will invoice FREIGHT VERIFY starting on Project start date (T) and will follow the payment schedule set below:

**Payment Terms:** Happiest Minds invoices will be raised at the end of each milestone and will be due for payment 30 days from the date of invoice.

## Terms and Conditions

**Travel:** If any travel and boarding/lodging related expenses are incurred during the execution of the project, it will be charged on actuals to FREIGHT VERIFY. Happiest Minds will seek prior permission from FREIGHT VERIFY before undertaking any such travel.

**Project specific Costs and Expenses:** Happiest Minds will provide standard PC hardware and software to its team members at its site for execution of work under this SOW. Any project specific specialized hardware, software licenses, testing devices or infrastructure required for the project will either be provided by FREIGHT VERIFY or will be procured and expensed to FREIGHT VERIFY. Happiest Minds will obtain prior written approval from FREIGHT VERIFY before procuring or incurring any project specific hardware, software, devices, or infrastructure expenditure.

**Taxes:** The pricing mentioned exclude GST and any other local and country specific taxes including any withholding tax, as may be applicable.

# About Happiest Minds Technologies limited

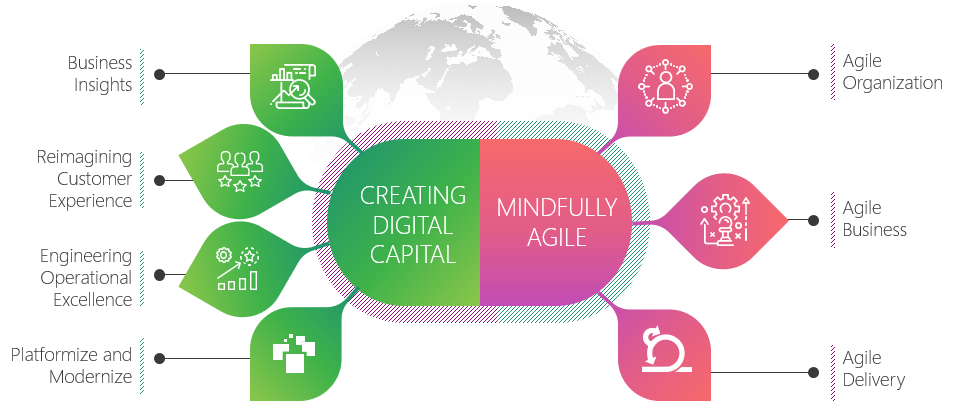
Happiest Minds Technologies Limited (NSE:HAPPSTMNDS) is a next generation digital transformation, infrastructure & security, and product engineering Services Company with 4500+ people, 16 locations and serving over 230+ customers.

Happiest Minds focus on digital transformation using technologies like IoT, Blockchain, Robotics, AR/VR etc. along with the nexus of forces of big data, analytics, cloud, and mobile gives it a unique position to help adoption of these technologies for our customers. The DNA of solution engineering with ability to create hardware/software/managed service solutions has been a key differentiator.

We were founded in Aug 2011 and within 9 years we have grown profitable had the one of the top 10 best IPOs in the last decade on September 9th, 2020.

We have a record of accomplishment of serving 230+ customers with +90% retention rate.

We consider ourselves “Born Digital. Born Agile”



We have been recognized as Great Place to work:

* **#29** India's Best Companies to Work for 2022
* **#68** Asia’s Best Workplaces 2022
* **Top 50** Best Workplaces for Women 2022
* India’s Top **15** Best Workplaces in Health & Wellness 2021
* Special Recognition for COVID-19 Support

With a mission of ‘Happiest People, Happiest Customers, we celebrate each of the “smilestones” with an act of giving. Meals are contributed to Akshaya Patra (mid-day meal scheme) in the name of our customers, our team, our business, our advisory board and Board members. .