# 1.2 AIB2.0 User Journey Workflows



## 1. Background & Context

AIBooster 2.0 aims to build a generic GenOps Platform on top of the existing AIB1.0 MLOps Platform while providing functional systems & templates to rapidly productionize a Generative AI business use case while maintaining best practices and encapsulating all possible aspects that an enterprise-grade platform should capture (scalability, robustness, logging, monitoring etc)

A developer leveraging this platform can adhere to multiple personas like an AIB ML Engineer, Use Case ML Engineer, AIB Platform Engineer etc.

This playbook aims to guide developers new to the platform through key user journeys to get started with the same in a simplified & concise manner.

| # | Git Repo | Description | Expected end Outcome |
|---|---|---|---|
| | | requirements while ensuring minimal impact to the process of productionizing a GenAI use case<br><br>This service is deployed as a Cloud Run instance in an AI Gateway Orchestration layer<br><br>For more details, please refer to 1.3.2 Responsible AI Service | the same can be used as a base to deploy the service to Cloud Run |
| 2 | LLM Proxy Service | This repository covers the logic running for the LLM Proxy Service.<br><br>Any LLM interactions that a GenAI use case needs to run passes through this service providing a framework to control which LLMs can a use case access & what kind of Responsible AI configuration is required for the same<br><br>For more details, please refer to 1.4.1 LLM Proxy Service | This repository outputs an image stored in a centrally located Google Cloud Artifact Registry via CI/CD &<br><br>the same can be used as a base to deploy the service to Cloud Run |
| 3 | AIB GenAI Template | This repository provides a reference implementation for productionizing a GenAI use case on Google Cloud, & can be used as a starting point w/ configurable boilerplate code. The implementation includes:<br>• Data ingestion and vectorisation pipeline using Kubeflow Pipelines and Vertex AI Pipelines.<br>• LLM inference using Vertex AI with RAG setup by Vertex AI Vector Search.<br>• Infrastructure-as-Code using Terraform for Cloud Run services, PubSub and Cloud Scheduler. | This repository once cloned for a specific use case, can output:<br>• A Base image to be used by the overall use case logic (RAG Inference + Data Vectorisation + LLM Evaluation)<br>• A Cloud Scheduler to run Data Vectorisation regularly<br>• A Cloud Scheduler to run Evaluation pipelines regularly |
| 4 | AIB Inference Deployments | This repository serves as a centrally-managed place to deploy any GenAI Use Case application as a Cloud Run Service.<br><br>Using YAML-based configurations hooked w/ automated CI/CD under-the-hood, a developer can quickly plug in configuration parameters to deploy an application across environments (lab/non-live/live) | This repository runs terraform-based deployment of all GenAI Use Case applications to Cloud Run via automated CI/CD under the hood |

# 3. Key User Journeys

Key User Journeys summarized below capture how different personas can leverage the 4 Git Repos mentioned above to successfully complete a relevant user journey

| # | | Persona | | User Journey | Applicable Git Repo |
|---|---|---|---|---|---|
| 1 | As a/an | AIB ML Engineer | ,I want to | Update the Responsible AI git repo | Responsible AI Service |
| 2 | | | | Add a new Responsible AI module | |
| 3 | | | | Deploy a new Responsible AI service image version | |
| 4 | | | | Update the Proxy service git repo | LLM Proxy Service |
| 5 | | | | Add a new Proxy router | |
| 6 | | | | Deploy a new Proxy service image version | |
| 7 | | Use Case ML Engineer/ Data Scientist | | Build an image for a GenAI use case | AIB GenAI Template |
| 8 | | | | Develop & test a Data Vectorisation pipeline | |
| 9 | | | | Create a Data Vectorisation pipeline schedule & Promote to higher environments | |
| 10 | | | | Develop & test the RAG inference logic | |
| 11 | | | | Deploy a RAG inference use case application & Promote to higher environments | AIB Inference Deployments |
| 12 | | | | Develop & Test an Evaluation pipeline - Golden Dataset pattern | AIB GenAI Template |
| 13 | | | | Develop & Test an Evaluation pipeline - Continuous evaluation pattern | |
| 14 | | | | Create an Evaluation pipeline schedule & Promote to higher environments | |
| 15 | | | | Update Proxy config for my use case | |

Workflows for these 15 User Journeys can be summarized in the designs below w/ further details on each User Journey captured in subsequent sub-pages:

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | | using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster |
| 2 | Add a python file for the new scanner | All Responsible AI modules exist in the `/responsible_ai_modules/scanners/` folder.<br>This includes existing modules for:<br>• Content Moderation using the Google Cloud Natural Language API<br>• Anonymization using the Google Cloud DLP API<br>• Prompt Injection using an open-source model deployed to a Vertex endpoint in the AI Gateway layer<br>New modules can include future roadmap functionality like topicality, embedding scans etc.<br><br>Any new module needs to exist in the above-mentioned folder w/ a base python class structure containing:<br>• The new scanner class deriving from the Base Scanner class<br>• 2 methods for:<br>  ○ base initialisation of the class<br>  ○ Scan method capturing underlying logic for the new module<br>Furthermore, all scanners/modules are generalized to adhere to a pydantic model called `IndividualScannerOutput` that exists in `responsible_ai_modules/schemas.py` . Any new module needs to transform their output to adhere to this structure | |
| 3 | Update supplementary files to include this new scanner | In addition to a new scanner file, following supplementary files need to be updated to include this new module:<br>• `responsible_ai_modules/scanners/__init__.py`<br>• `get_scanner_by_name` function in `responsible_ai_modules/scanners/utils.py` | |
| 4 | Add unit tests | The repo contains unit tests related to all existing functionality in the `tests/` folder | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | This markdown file includes further details on testing the service locally & miscellaneous related information on unit testing etc | Notebook - AI Booster<br>• VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster |
| 2 | Refactor code as required | Once the development workspace has been completely setup, a developer can create a feature branch, refactor code as required & commit changes to said feature branch | |
| 3 | Update existing unit tests | The repo contains unit tests related to all existing functionality in the `tests/` folder<br>These tests can be updated based on the refactoring of functionality as required & tested locally by executing the `make pytest` command.<br>This Makefile abstracts away execution of key commands & can be referred for other aspects as well | |
| 4 | Test changes locally | Once required changes have been committed, the same can be tested locally by spinning up the LLM Proxy Service locally using `make run` which spins up the FastAPI server on localhost:8000 by default<br>Local testing can be executed in 2 ways:<br>• `curl` directly into the local server<br>• Spin up a dummy use case app locally using the `make run-dummy-use-case` command & update the `dummy_use_case/api/app.py` file as required<br>The 2nd option provides more flexibility to test any LLM Proxy changes from an overall usage standpoint (calling the LLM Proxy via Langchain, Llama-Index or other LLM frameworks) | |
| 5 | Create a Pull request | Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review | |
| 6 | Merge to main | Post review & approval of the feature branch functionality, the same can be merged into the `main` branch | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | New routers can include LLMs that do not conform to the extensibility provided by above-mentioned routers. In that scenario, a new router file needs to be added to this folder w/ relevant authentication logic & API calling logic.<br><br>Please refer to existing Google LLM, External LLM routers for ideas on standardized code structure<br><br>Once logic for the new router has been plugged in, this router needs to be included in the base `api/app.py` file | |
| 3 | Add logic for parsing relevant new model patterns | Every LLM call via a router conforms to a generalized regex model pattern that captures all possible model paths as per the final LLM endpoint. This is currently captured in the `env.sh` file & the same is deployed with LLM Proxy Cloud Run deployment as environment variables.<br><br>In addition to the above, 2 utility files capture logic to generically parse these model patterns & the same need to be amended:<br>• api/utils/model_path_parsing.py - Need to add a new case for the new model patterns<br>• api/utils/schema.py - Need to update the `ModelProvider` class to reflect this new model pattern | |
| 4 | Add unit tests | The repo contains unit tests related to all existing functionality in the `tests/` folder<br><br>This folder will need to be updated w/ new unit tests to reflect new functionality created for the new scanner.<br><br>This can be tested locally by executing the `make pytest` command.<br><br>This Makefile abstracts away execution of key commands & can be referred for other aspects as well | |
| 5 | Test changes locally | Once required changes have been committed, the same can be tested locally by spinning up the LLM Proxy Service locally using `make run` which spins up the FastAPI server on localhost:8000 by default<br><br>Local testing can be executed in 2 ways:<br>• `curl` directly into the local server<br>• Spin up a dummy use case app locally using the `make run-dummy-use-case` command & update the `dummy_use_case/api/app.py` file as required | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| 3a | CI process automatically builds & pushes image to Dev Booster Mirror | Once a tag w/ dev as suffix is created, the CI pipeline is automatically triggered that runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of Dev Booster Mirror<br><br>For more details on this CI pipeline run by Cloud Build, please refer to the `Dockerfile` & `cloudbuild.yaml` files in the repo | Cloud Build Trigger exists in the Dev Booster Mirror Google Cloud project to run this CI pipeline.<br><br>This is a manual one-time activity & is already in place |
| 3b | CI process automatically builds & pushes image to Prod Booster Mirror | Once a tag w/ prd as suffix is created, the CI pipeline is automatically triggered that runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of Prod Booster Mirror<br><br>For more details on this CI pipeline run by Cloud Build, please refer to the `Dockerfile` & `cloudbuild.yaml` files in the repo | Cloud Build Trigger exists in the Prod Booster Mirror Google Cloud project to run this CI pipeline.<br><br>This is a manual one-time activity & is already in place |

# 1.2.2 User Journeys - Persona: Use Case ML Engineer/ Data Scientist



## 1. Persona Description

As a Use Case ML Engineer or a Use Case Data Scientist, key responsibilities can include leveraging the AIB GenAI Template as boilerplate code to build out relevant logic for a specific GenAI use case & productionize the same w/ components provided by the AIB2.0 Platform.

These templates capture the following essential elements of a GenAI use case:

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | as required & commit changes to said feature branch<br><br>The repo also contains unit tests related to all existing functionality in the `testing/` folder<br><br>These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command.<br><br>This Makefile abstracts away execution of key commands & can be referred for other aspects as well | |
| 4 | Create a pull request | Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review | |
| 5a | Merge to main | Post review & approval of the feature branch functionality, the same can be merged into the `main` branch | |
| 5b | Create a Release tag | A new release tag needs to be created w/ relevant description supporting it.<br><br>This tag should conform to best practices around tags, specifically of format:<br><br>vMAJOR.MINOR.PATCH<br><br>eg - A new release tag can be:<br>• v1.2.0 | |
| 6a | CI process automatically builds & pushes an image to the build-nl Use Case Vendor layer project | Once the PR has been pushed to the `main` branch (see step 5a), the CI/CD pipeline is automatically triggered that:<br>• Runs relevant unit testing & linting checks<br>• Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-nl project with a `latest` tag along with the relevant git commit SHA value<br>• Compiles relevant Vertex pipelines & stores them in GCS (more details to follow in subsequent user journeys)<br>• Create/update a Cloud Scheduler job for the compiled Vertex pipeline (more details to follow in subsequent user journeys) | Cloud Build Trigger exists in the Use Case Vendor layer build-nl project to run this CI/CD pipeline<br><br>This is a one-time manual activity & needs to be implemented once for every GenAI use case |

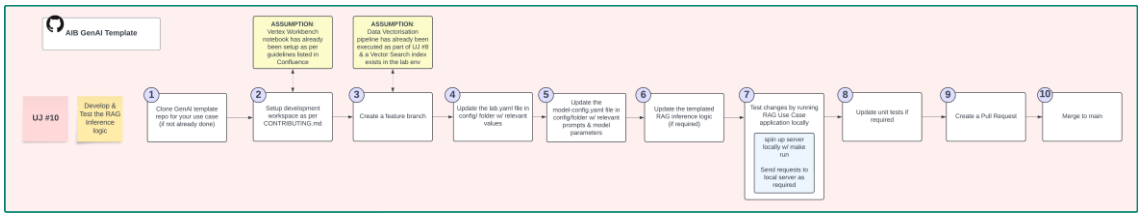
| # | Element | Description | Assumptions |
|---|---|---|---|
| | | | Notebook - AI Booster<br>• VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster |
| 3 | Create a feature branch | Create a feature branch on the cloned repo to build & test relevant Data Vectorisation pipeline functionality | |
| 4 | Update the lab.yaml file in config/ w/ relevant values | In addition to boilerplate code to build a Vertex pipeline for Data Vectorisation & ingest documents to a Vector Database (Vertex Vector Search), this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder<br>This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances<br>For testing purposes, the lab.yaml file needs to be updated w/ relevant values catering to:<br>• Project ID<br>• Region<br>• Relevant Google Cloud Storage URIs<br>• Relevant Vertex Pipeline configuration<br>• Vertex Vector Search fields<br>The template repo contains sample YAML files for reference | Source data files to be ingested into a Vector Database already exist in a relevant Google Cloud Storage bucket in the relevant Use Case Vendor layer project |
| 5 | Update code for Vertex pipeline components & the pipeline (if required) | Once the configuration-based YAML files have been updated, base logic in the `ingestion_workflow/pipeline/` folder can be updated if absolutely required.<br>This folder contains generic logic for Vertex Pipeline components that include:<br>• Fetching documents<br>• Processing documents | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | • Creating or Updating a Vertex Vector Search index (Vector Database)<br>• Updating pipeline details<br>All components above contain core logic to maintain & audit the document lifecycle of processed documents | |
| 6 | Update unit tests (if required) | The repo contains unit tests related to all existing functionality in the `testing/` folder<br>These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command.<br>This Makefile abstracts away execution of key commands & can be referred for other aspects as well | |
| 7 | Manually compile & submit a Vertex Pipeline job | Once all changes are in place, the Vertex pipeline can be manually submitted via a single python command - `python ingestion_workflow/pipeline/pipeline_run.py` to test updated functionality | |
| 8a | Validate Vertex Pipeline execution | Once the above-mentioned python command has been executed in a development workspace, the Terminal will provide a link for the running pipeline OR the same can be found in the Vertex Pipeline UI<br>Once submitted & executed, this pipeline should be validated via the UI to verify updated functionality & configuration (if any) | |
| 8b | Validate create Vector Database | Once the Vertex pipeline has been submitted & successfully executed, a Vertex Vector Search index will be created/updated w/ embeddings of the ingested documents.<br>The same should be validated from a use case specific standpoint | |
| 9 | Create a Pull Request | Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review | |
| 10 | Merge to main | Post review & approval of the feature branch functionality, the same can be merged into the `main` branch | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| 3 | Update the cron schedule for the Cloud Scheduler job | In addition to relevant configuration-based YAML files, separate files contain logic to configure the cron schedule of the Cloud Scheduler job to be created/updated<br><br>cron schedules for both non-live & live Use Case Vendor layer projects will need to be updated separately | |
| 4 | Create a Pull Request | Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review | |
| 5a | Merge to main | Post review & approval of the config-update/feature branch functionality, the same can be merged into the `main` branch | |
| 5b | Create a Release tag | A new release tag needs to be created w/ relevant description supporting it.<br><br>This tag should conform to best practices around tags, specifically of format:<br><br>vMAJOR.MINOR.PATCH<br><br>eg - A new release tag can be:<br>• v1.2.0 | |
| 6a | CI process automatically creates/updates the Cloud Scheduler job in the non-live Use Case Vendor layer project | Once the PR has been pushed to the `main` branch (see step 5a), the CI/CD pipeline is automatically triggered that:<br>• Runs relevant unit testing & linting checks<br>• Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-nl project with a `latest` tag along with the relevant git commit SHA value<br>• Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the git commit SHA value in the build-nl Use Case Vendor layer project<br>• Create/update a Cloud Scheduler job for the compiled Vertex pipeline in the non-live Use Case Vendor layer project<br>  ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the | Cloud Build Trigger exists in the Use Case Vendor layer build-nl project to run this CI/CD pipeline<br><br>This is a one-time manual activity & needs to be implemented once for every GenAI use case |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | git commit SHA value attached to the image + the compiled pipeline GCS location from the previous stage | |
| 6b | CI process automatically creates/updates the Cloud Scheduler job in the live Use Case Vendor layer project | Once a new release tag has been created (see step 5b), the CI/CD pipeline is automatically triggered that:<br>• Runs relevant unit testing & linting checks<br>• Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-live project with the release tag attached to the image<br>• Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the release tag in the build-live Use Case Vendor layer project<br>• Create/update a Cloud Scheduler job for the compiled Vertex pipeline in the live Use Case Vendor layer project<br>  ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the release tag attached to the image + the compiled pipeline GCS location from the previous stage | Cloud Build Trigger exists in the Use Case Vendor layer build-live project to run this CI/CD pipeline<br>This is a one-time manual activity & needs to be implemented once for every GenAI use case |

## 2.4. User Journey #10 - Develop & Test the RAG Inference logic




Based on the user journey outlined above, this UJ applies to the AIB GenAI Template git repo & details for the same can be broken down as follows:

| # | Element | Description | Assumptions |
|---|---|---|---|
| 1 | Clone GenAI Template repo | Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| 2 | Setup a development workspace | Once the assumptions have been met, further setup is required. This includes:<br>• Python package installation using poetry as a package manager<br>• Loading environment variables<br>Detailed steps for the same are captured in the CONTRIBUTING.md of the repo | Development workspace has been setup by:<br>• Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster<br>• VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster |
| 3 | Create a feature branch | Create a feature branch on the cloned repo to build & test relevant RAG inference logic | Data Vectorisation pipeline has already been successfully executed as part of UJ #8 &<br>a Vector Search index exists containing embeddings of the ingested documents |
| 4 | Update the lab.yaml file in config/ w/ relevant values | In addition to boilerplate FastAPI code to build out RAG inference logic, this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder<br>This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances<br>For testing purposes, the lab.yaml file needs to be updated w/ relevant values catering to:<br>• Project ID<br>• Region<br>• Relevant Google Cloud Storage URIs<br>• Relevant Vertex Pipeline configuration<br>• Vertex Vector Search fields<br>The template repo contains sample YAML files for reference | |
| 5 | Update the model-config.yaml file | Furthermore, for inference purposes, a separate model-config YAML file captures inference-based configurations. This includes: | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | in config/ w/ relevant values | • LLM parameters around temperature, top_k, top_p etc<br>• LLM Prompts as YAML fields<br>• ...<br>The template repo contains sample YAML files for reference | |
| 6 | Update the templated RAG inference logic | This template repo contains generic code to run a single LLM as part of its logic<br>If a use case requires multiple LLM calls or any additional processing (reranking etc), the same needs to be reflected in the:<br>• inference_workflow/src/ folder (contains source scripts for inference & retrieval)<br>• inference_workflow/api/routers/ folder (contains FastAPI logic which imports source scripts & executes them in a predict router | |
| 7 | Test changes by running locally | Once required changes have been committed, the same can be tested locally by spinning up the Use Case Application locally using `make run` which spins up the FastAPI server on localhost:8000 by default<br>Local testing can be executed by running a `curl` command to send API calls directly into the local server | |
| 8 | Update unit tests if required | The repo contains unit tests related to all existing functionality in the `testing/` folder<br>These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command.<br>This Makefile abstracts away execution of key commands & can be referred for other aspects as well | |
| 9 | Create a Pull Request | Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review | |
| 10 | Merge to main | Post review & approval of the feature branch functionality, the same can be merged into the `main` branch | |

## 2.5. User Journey #11 - Deploy a RAG Inference Use Case Application & Promote to higher environments

| # | Element | Description | Assumptions |
|---|---|---|---|
| | w/ evaluation-based values | • LLM evaluation on a pre-set Golden Dataset (prompt + expected response)<br>• Continuous LLM evaluation based on actual prompt-response pairs generated by the RAG Use Case Application<br>In addition to this generic code to build Vertex pipelines for the above-mentioned 2 patterns, this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder<br>This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances<br>For testing purposes, the relevant YAML file needs to be updated w/ evaluation-specific values catering to:<br>• Project ID<br>• Region<br>• Evaluation metrics<br>• Input Golden Dataset BQ location<br>• Expected output BQ location to store evaluation results<br>The template repo contains sample YAML files for reference | pairs already exist in a BQ table as a one-time activity |
| 5 | Update code for Vertex pipeline components & the pipeline (if required) | Once the configuration-based YAML files have been updated, base logic in the `evaluation/pipeline/` folder can be updated if absolutely required.<br>This folder contains generic logic for Vertex Pipeline components that include:<br>1. Extracting data from a BQ table<br>2. Querying a Use Case app to generate relevant responses<br>3. Running LLM Evaluation using the Google GenAI Evaluation service<br>4. Storing evaluation results in the relevant output BQ table<br>For the Golden Dataset Evaluation pattern, all 4 components apply & have been plugged into a generic Vertex pipeline | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| 1 | Clone GenAI Template repo | Clone the template repo for your GenAI use case & leverage the boilerplate code & frameworks already provided by the template | |
| 2 | Setup a development workspace | Once the assumptions have been met, further setup is required. This includes:<br>• Python package installation using poetry as a package manager<br>• Loading environment variables<br>Detailed steps for the same are captured in the CONTRIBUTING.md of the repo | Development workspace has been setup by:<br>• Creating a Vertex Workbench instance using this guide - How to : Create a Vertex Notebook - AI Booster<br>• VSCode has been connected to the created Workbench instance using this guide - How to : Connect to AI Booster Notebook vis SSH - AI Booster |
| 3 | Create a feature branch | Create a feature branch on the cloned repo to build & test relevant LLM Evaluation - Continuous Evaluation functionality | |
| 4 | Update the relevant YAML file/s in config/ w/ evaluation-based values | This template repo contains generic code to execute LLM evaluation catering to 2 patterns:<br>• LLM evaluation on a pre-set Golden Dataset (prompt + expected response)<br>• Continuous LLM evaluation based on actual prompt-response pairs generated by the RAG Use Case Application<br>In addition to this generic code to build Vertex pipelines for the above-mentioned 2 patterns, this template abstracts relevant updates to a configuration-based YAML file which exists in the **config/** folder<br>This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances<br>For testing purposes, the relevant YAML file needs to be updated w/ evaluation-specific values catering to:<br>• Project ID<br>• Region | • Use Case application has been created as part of UJ #10 + #11 & deployed as a Cloud run instance<br>• BigQuery log sink has been created for the Use Case Inference Cloud Run application & contains information on the actual prompt-response pairs generated by the app |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | • Evaluation metrics<br>• Input BQ location containing actual prompt-response pairs generated by a Use Case Application<br>• Expected output BQ location to store evaluation results<br>The template repo contains sample YAML files for reference | |
| 5 | Update code for Vertex pipeline components & the pipeline (if required) | Once the configuration-based YAML files have been updated, base logic in the `evaluation/pipeline/` folder can be updated if absolutely required.<br>This folder contains generic logic for Vertex Pipeline components that include:<br>1. Extracting data from a BQ table<br>2. Querying a Use Case app to generate relevant responses<br>3. Running LLM Evaluation using the Google GenAI Evaluation service<br>4. Storing evaluation results in the relevant output BQ table<br>For the Continuous Evaluation pattern, components 1,3,4 apply & have been plugged into a generic Vertex pipeline | |
| 6 | Update unit tests (if required) | The repo contains unit tests related to all existing functionality in the `testing/` folder<br>These tests can be updated based on the refactoring as required & tested locally by executing the `make pytest` command.<br>This Makefile abstracts away execution of key commands & can be referred for other aspects as well | |
| 7 | Manually compile & submit a Vertex Pipeline job | Once all changes are in place, the Vertex pipeline can be manually submitted via a single python command to test updated functionality | |
| 8a | Validate Vertex Pipeline execution | Once the above-mentioned python command has been executed in a development workspace, the Terminal will provide a link for the running pipeline OR the same can be found in the Vertex Pipeline UI<br>Once submitted & executed, this pipeline should be validated via the UI to verify updated functionality & configuration (if any) | |
| 8b | Validate output BQ table | Once the Vertex pipeline has been submitted & successfully executed, an | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | configuration-based YAML file which exists in the **config/** folder<br><br>This config folder will need a sub-folder w/ naming of the AIB instance (eg vf-uk-aib-prd-cip-ask-lab/nl/live points to vf-uk-aib-prd-cip-ask) & YAML files within said sub-folder pointing to the lab, non-live & live instances<br><br>For promotion purposes, the relevant YAML files for non-live & live need to be updated w/ evaluation-specific values catering to:<br>• Project ID<br>• Region<br>• Evaluation metrics<br>• Input BQ location containing actual prompt-response pairs generated by a Use Case Application<br>• Expected output BQ location to store evaluation results<br><br>The template repo contains sample YAML files for reference<br><br>Additionally, configuration values setup in the lab YAML as part of pipeline testing in UJ #13 should be used as a reference | |
| 3 | Update the cron schedule for the Cloud Scheduler job | In addition to relevant configuration-based YAML files, separate files contain logic to configure the cron schedule of the Cloud Scheduler job to be created/updated<br><br>cron schedules for both non-live & live Use Case Vendor layer projects will need to be updated separately | |
| 4 | Create a Pull Request | Once relevant changes have been tested & the feature branch is good to go, next logical step would be to create a Pull Request in the repo for review | |
| 5a | Merge to main | Post review & approval of the config-update/feature branch functionality, the same can be merged into the `main` branch | |
| 5b | Create a Release tag | A new release tag needs to be created w/ relevant description supporting it. | |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | This tag should conform to best practices around tags, specifically of format:<br>vMAJOR.MINOR.PATCH<br>eg - A new release tag can be:<br>• v1.2.0 | |
| 6a | CI process automatically creates/updates the Cloud Scheduler job in the non-live Use Case Vendor layer project | Once the PR has been pushed to the `main` branch (see step 5a), the CI/CD pipeline is automatically triggered that:<br>• Runs relevant unit testing & linting checks<br>• Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-nl project with a `latest` tag along with the relevant git commit SHA value<br>• Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the git commit SHA value in the build-nl Use Case Vendor layer project<br>• Create/update a Cloud Scheduler job for the compiled Vertex pipeline/s in the non-live Use Case Vendor layer project<br>  ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the git commit SHA value attached to the image + the compiled pipeline GCS location from the previous stage | Cloud Build Trigger exists in the Use Case Vendor layer build-nl project to run this CI/CD pipeline<br>This is a one-time manual activity & needs to be implemented once for every GenAI use case |
| 6b | CI process automatically creates/updates the Cloud Scheduler job in the live Use Case Vendor layer project | Once a new release tag has been created (see step 5b), the CI/CD pipeline is automatically triggered that:<br>• Runs relevant unit testing & linting checks<br>• Runs the image build & push process & finally outputs an image in Google Cloud Artifact registry of the Use Case Vendor layer build-live project | Cloud Build Trigger exists in the Use Case Vendor layer build-live project to run this CI/CD pipeline<br>This is a one-time manual activity & needs to be implemented once for every GenAI use case |

| # | Element | Description | Assumptions |
|---|---|---|---|
| | | with the release tag attached to the image<br>• Compiles relevant Vertex pipelines & stores them in GCS under a folder w/ name mapping to the release tag in the build-live Use Case Vendor layer project<br>• Create/update a Cloud Scheduler job for the compiled Vertex pipeline/s in the live Use Case Vendor layer project<br>  ○ This is done via Terraform embedded into the CI/CD pipeline as its final stage & it utilizes the release tag attached to the image + the compiled pipeline GCS location from the previous stage | |

# 1.2.3 User Journeys - Proxy Config Update



## 1. User Journey-specific Background & Context

## 2.1. User Journey #15 - Update Proxy Config for my use case (Current Approach)