MLOps Strategy & Framework

Phase 1 Playback

March 23rd, 2023

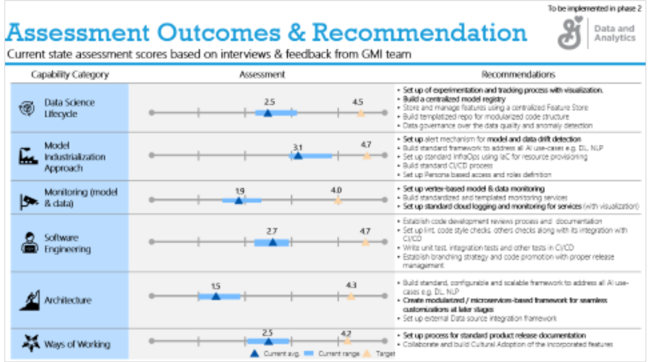
We are finalizing Phase 1 and will launch implementation next week

	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12
Mobilization				Today								
Access GMI environment	■											
Journey to state-of-the-art MLOps framework												
Review GMI assessment and gap to explore	■											
Review current processes/workflows	■	■										
Define capabiltiies to build and architecture / tooling	■	■	■									
Build detailed implementation plan		■	■	▨	▨	▨	▨	▨	▨	▨	▨	■
Define scope and prioritization for implementation			■ ◆									
Implementation of roadmap on Walmart Engine												
Review code (partial team)	■	■	■									
Implement MLOps capabilities for use case		▨	▨	■	■ ↻	■	■ ↻	■	■ ↻	■	■	■ ↻
Best practices Documentation												
Develop documentation to implement MLOps to use case											■	■
	▲ Kick off			▲ SC #1								▲ SC #2

◆ Workshops ▲ Key meetings ↻ Sprint review

We collaborated with ML Team to understand current capabilities and roadmap

Assessment



To be implemented in phase 2

Assessment Outcomes & Recommendation

Data and Analytics

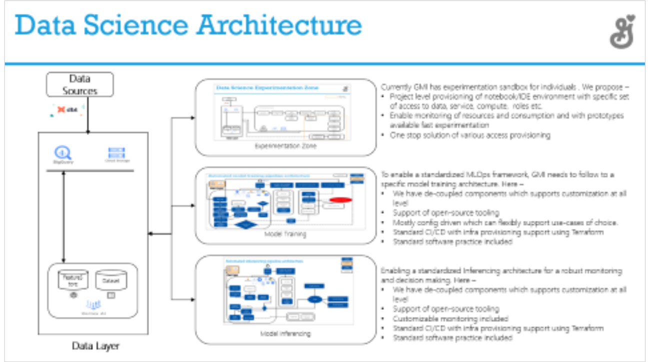
Current state assessment scores based on interviews & feedback from GMI team

Capability Category	Assessment	Recommendations
Data Science Lifecycle	2.5 4.5	• Set up of experimentation and tracking process with visualization. • Build a centralized model registry • Store and manage features using a centralized Feature Store • Build templatized repo for modularized code structure • Data governance over the data quality and anomaly detection
Model Industrialization Approach	3.1 4.7	• Set up alert mechanism for model and data drift detection • Build standard framework to address all AI use-cases e.g. DL, NLP • Set up standard InfraOps using IaC for resource provisioning • Build standard CI/CD process • Set up Persona based access and roles definition
Monitoring (model & data)	1.9 4.0	• Set up vertex-based model & data monitoring • Build standardized and templated monitoring services • Set up standard cloud logging and monitoring for services (with visualization)
Software Engineering	2.7 4.7	• Establish code development reviews process and documentation • Set up lint, code style checks, others checks along with its integration with CI/CD • Write unit test, integration tests and other tests in CI/CD • Establish branching strategy and code promotion with proper release management
Architecture	1.5 4.3	• Build standard, configurable and scalable framework to address all AI use-cases e.g. DL, NLP • Create modularized / microservices-based framework for seamless customizations at later stages • Set up external Data source integration framework
Ways of Working	2.5 4.2	• Set up process for standard product release documentation • Collaborate and build Cultural Adoption of the incorporated features

▲ Current avg. ▬ Current range △ Target

- Leveraged ACN framework to assess GMI MLOps capabilities at overall and project level
- Received 7 assessment responses which were validated
- Based on the scores provided suggested for a futuristic target score which could be achieved with provided recommendation

Interviews / Workshops



Data Science Architecture

Currently GMI has experimentation sandbox for individuals . We propose –

- Project level provisioning of notebook/IDE environment with specific set of access to data, service, compute, roles etc
- Enable monitoring of resources and consumption and with prototypes available fast experimentation
- One stop solution of various access provisioning

To enable a standardized MLOps framework, GMI needs to follow to a specific model training architecture. Here –

- We have de-coupled components which supports customization at all level
- Support of open-source tooling
- Mostly config driven which can flexibly support use-cases of choice.
- Standard CI/CD with infra provisioning support using Terraform
- Standard software practice included

Enabling a standardized Inferencing architecture for a robust monitoring and decision making. Here –

- We have de-coupled components which supports customization at all level
- Support of open-source tooling
- Customizable monitoring included
- Standard CI/CD with infra provisioning support using Terraform
- Standard software practice included

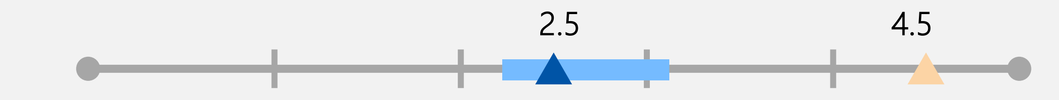
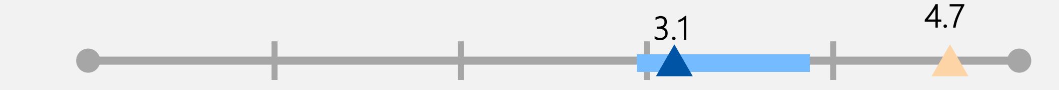
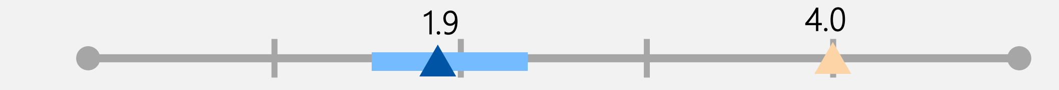
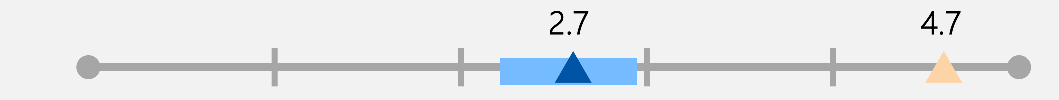
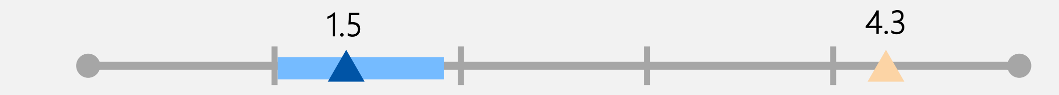
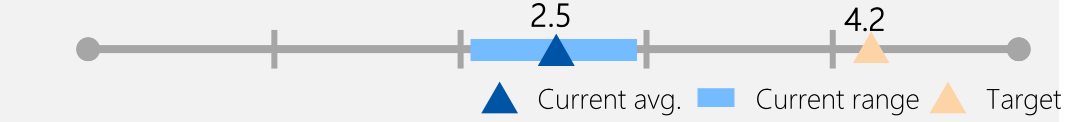
- Conducted individual interviews to understand maturity of MLOPS framework at GMI
- Multiple session on Architecture and solution design
- Sessions on SRM Walmart use-case
- Reviewed the recommendations provided by ACN Teams and accommodated GMI current plan and practise. Redefined ACN recommendation

To be implemented in phase 2

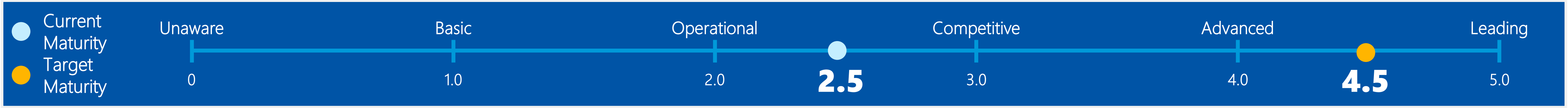
Data and Analytics

Assessment Outcomes & Recommendation

Current state assessment scores based on interviews & feedback from GMI team

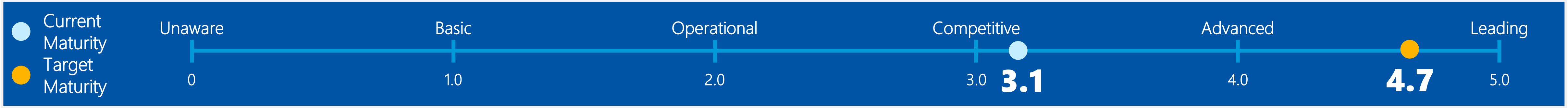
Capability Category	Assessment	Recommendations
Data Science Lifecycle		• **Set up of experimentation and tracking process with visualization.** • **Build a centralized model registry** • Store and manage features using a centralized Feature Store • Build templatized repo for modularized code structure • Data governance over the data quality and anomaly detection
Model Industrialization Approach		• **Set up** alert mechanism for **model and data drift detection** • Build standard framework to address all AI use-cases e.g. DL, NLP • Set up standard InfraOps using IaC for resource provisioning • Build standard CI/CD process • Set up Persona based access and roles definition
Monitoring (model & data)		• **Set up vertex-based model & data monitoring** • Build standardized and templated monitoring services • **Set up standard cloud logging and monitoring for services** (with visualization)
Software Engineering		• Establish code development reviews process and documentation • Set up lint, code style checks, others checks along with its integration with CI/CD • Write unit test, integration tests and other tests in CI/CD • Establish branching strategy and code promotion with proper release management
Architecture		• Build standard, configurable and scalable framework to address all AI use-cases e.g. DL, NLP • **Create modularized / microservices-based framework for seamless customizations at later stages** • Set up external Data source integration framework
Ways of Working		• **Set up process for standard product release documentation** • Collaborate and build Cultural Adoption of the incorporated features

Data Science Lifecycle



	Successes/Observations	Opportunities
Data Pipelines	• Data pipelines for ingestion do some quality checks and validations • Historical version data in place for model tables • Alation used to store data/model definitions • Features have localized landing tables per-project & devs have process in-place to work with them for experimentation	• Centralize model feature storage and definitions in Vertex Feature Store to facilitate monitoring, reusability, and streamline ingestion & serving • Incorporate input data versioning to enable lineage tracking
Model Development	• Template repo is available to bootstrap new analytics projects • Well defined GitFlow branching strategy in-place with CICD workflows covering dev -> prod	• Automate experiment tracking during model development using Vertex Experiments and standardized doc templates • Setup infra & processes to allow sharing reusable code packages/components amongst project teams
Ideation & Pipeline Management	• Walmart Engine is working towards being an example of the reference architecture internally • There are pre-defined gates for business to evaluate if project is meeting business goals	• Standardized project checklist/intake that clearly identifies data requirements, upstream/downstream integrations, and defined business KPIs

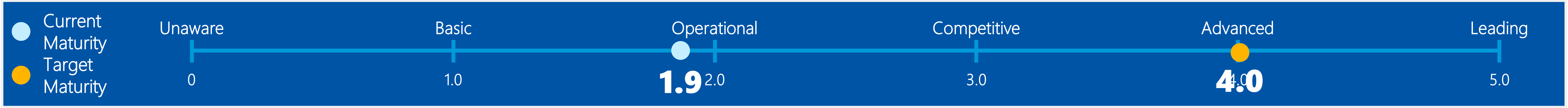
Model Industrialization Approach



● Current Maturity
● Target Maturity

	Successes/Observations	Opportunities
Functional Requirements	• Development process is cloud-based with Vertex Notebooks • Models are tracked in a model registry prior to deployment • Model training and serving occurs via orchestrated pipeline(s)	• Component and pipeline-based model training and serving to improve code reusability and follow standard • Incorporate model and data drift detection with alert mechanisms into training and serving pipelines • Enable automated model retrain and promotion capabilities from pipelines
Non-Functional Requirements	• Already leveraging auto-scaling, serverless environments (BigQuery/Vertex Endpoints) to minimize infrastructure management concerns (scalability/DR) • Logging and monitoring is available in some capacity in most stages of model development & deployment • Role-based access to data	• Leverage managed services and serverless environments when possible (e.g. Composer -> Vertex Pipelines) • Create production checklist for deployment and run-book for post-deployment • Document persona-based access and roles definition and establish access guidelines for data (data sensitivity, AD groups, request method, etc.)

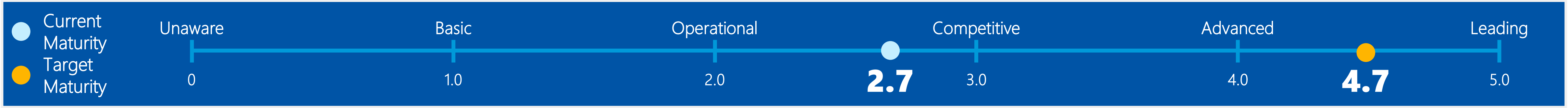
Monitoring



	Successes/Observations	Opportunities
Model Performance	• Model performance can be visualized with model card dashboard and is integrated into pipelines • Business validation of models occurs prior to production deployment	• Formalize model availability using SLOs that track system health metrics (throughput, error rate, latency etc.) as well as model-specific • Tie model metrics to business KPIs (directly or by proxy) to help business understand model performance in relation to business impact and quantify model ROI • Establish and document model performance thresholds that can be alerted on
Fairness Monitoring	• Dev team and business have a shared understanding of model and impact of model predictions on downstream use	• Document potential sources of bias for each model and have automated checks in place for these when possible • Leverage model prediction explanation tools (Feature attribution)
Continuous Training	• Model training and deployment is orchestrated via trigger-able pipelines • Model deployment has staged release and rollback is possible	• Modify training pipeline to support scheduled or event-driven (new data, performance degradation) invocations • Allow for long-term model performance tracking • Document feedback loops for model- how will model's output be correlated to future input data?

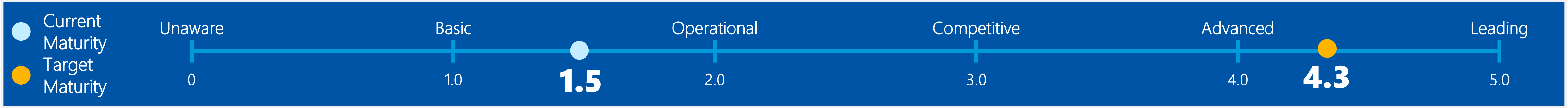
Software Engineering

Data and Analytics



	Successes/Observations	Opportunities
Code Best Practices	• Local linting and formatting during development • Code has review process prior to merge in dev/main • Some automated integration testing	• Define target test coverage (e.g. 70-90%) for production code • Enhance CI workflow to incorporate code quality checks (linting, formatting, type-checking, docstrings, etc.) that prevent merge unless all passed • Improve code readability and consistency across projects by defining or adopting a style-guide convention
DevOps Best Practices	• There is a well-defined branching strategy (GitFlow) and code promotion with proper release management • Formal process in place for release management	• Ensure test suite is covering unit, functional, and integration testing • Create a standardized production-readiness checklist

Architecture

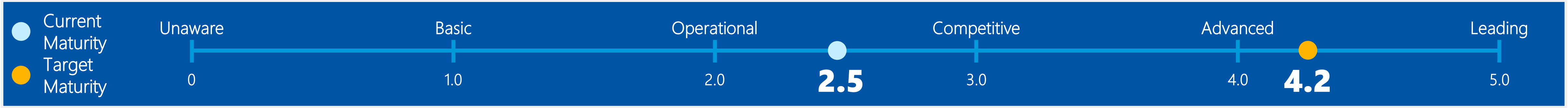


● Current Maturity
● Target Maturity

Unaware 0 — Basic 1.0 — **1.5** — Operational 2.0 — Competitive 3.0 — Advanced 4.0 — **4.3** — Leading 5.0

	Successes/Observations	Opportunities
Cloud Architecture	• Separate cloud environments for ML project exist with identical configurations • Code and artifacts are deployable to each environment	• Move towards fully-reproducible environments via IaC that require little to no manual steps to setup/teardown ("Project Factory" module) • Ensure production Endpoints and resources are fully isolated from lower environments
Data Requirements	• Data for models comes from centralized EDW • Role-based access to data based on AD groups • Some data quality checks and analysis occurs at ingest	• Standardize logging and monitoring of data transforms • Monitor and alert on upstream data quality issues with remediation plan & contacts in place • Leverage Vertex Metadata to version data artifacts and store metadata about data used (owners, fields, sensitivity, etc.)

Ways of Working



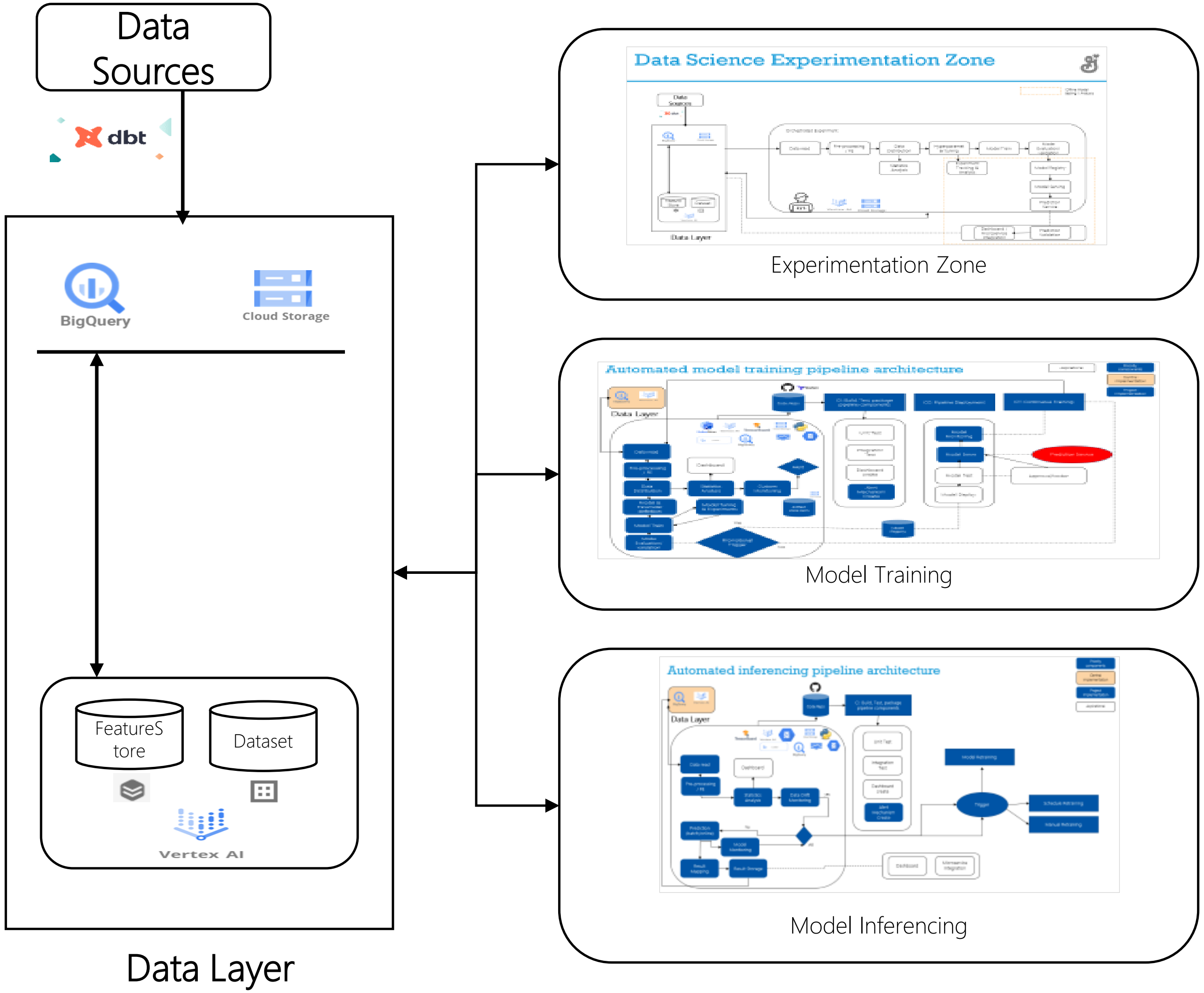
	Successes/Observations	Opportunities
Upskilling & Knowledge Transfer	• Some documentation in place for projects with partial templates for generation • Relevant training and upskilling in MLOps available for MLEs	• Create standard documentation templates for on-boarding to a project • Create standardized documentation templates for models that can be linked to via model registry/metadata
Team Skills and Structure	• Roles on analytics projects are mostly defined and consistent across projects	• Formalize roles on project and incorporate into project documentation with relevant points of contact for business, engineering, data science, etc.
Relationships and Communication	• Business is involved in model validation process and model's usage is not black-box to engineers	• Add mappings of all upstream and downstream integration points to project documentation

Priority 1 (week 5-9) implementation scope items

Actions	Details	Deliverables
Set up experimentation & Tracking process with Model Registry	• Develop Kubeflow components for Training pipeline. Components such as data read, data process, data distribution, statistics generation • Components such as model training, hyperparameter tuning, deployment and serving • Pipeline code development in Kubeflow • Develop Kubeflow code and set up experimentation • Model registry set up on vertex • Training pipeline components	• Kubeflow based components (code) • Kubeflow pipeline (code) • Model Registry set up
Set up model registry on Vertex	• Develop Kubeflow code and set up experimentation • Model registry set up on vertex • Training pipeline components	• Kubeflow based components (code) • Kubeflow pipeline (code) • Model Registry set up
Model and Data Drift detection	• Log based monitoring/alerting solutions, if any • Create cloud function scripts to trigger re-training • Create cloud scheduler script to trigger re-training	• Cloud logging • Custom log based monitoring/alerting to be set up on GCP. • Custom script development (as required).
Set up vertex-based model & data monitoring	• Use Vertex to set up model and data drift detection	• Custom code development in Kubeflow/python for drift detection using vertex SDK
Cloud Logging & Monitoring of Services	• Set up Monitoring dashboard for services • Cloud logging throughout the project	• Cloud logging • Monitoring Dashboard (as required)
Create modularized / microservices-based framework for seamless customizations at later stages	• Standard code design patterns and make solutions to seamlessly address custom requirements in future	• Custom scripts (as required)
Set up process for standard product release documentation	• Establish code development reviews process • Prepare documentation • Knowledge Transfer sessions	• Document • Sessions

We will work in sprints and deliver additional capabilities as time allows

Data Science Architecture



Currently GMI has experimentation sandbox for individuals . We propose –

- Project level provisioning of notebook/IDE environment with specific set of access to data, service, compute, roles etc.
- Enable monitoring of resources and consumption and with prototypes available fast experimentation
- One stop solution of various access provisioning

To enable a standardized MLOps framework, GMI needs to follow to a specific model training architecture. Here –

- We have de-coupled components which supports customization at all level
- Support of open-source tooling
- Mostly config driven which can flexibly support use-cases of choice.
- Standard CI/CD with infra provisioning support using Terraform
- Standard software practice included

Enabling a standardized Inferencing architecture for a robust monitoring and decision making. Here –

- We have de-coupled components which supports customization at all level
- Support of open-source tooling
- Customizable monitoring included
- Standard CI/CD with infra provisioning support using Terraform
- Standard software practice included

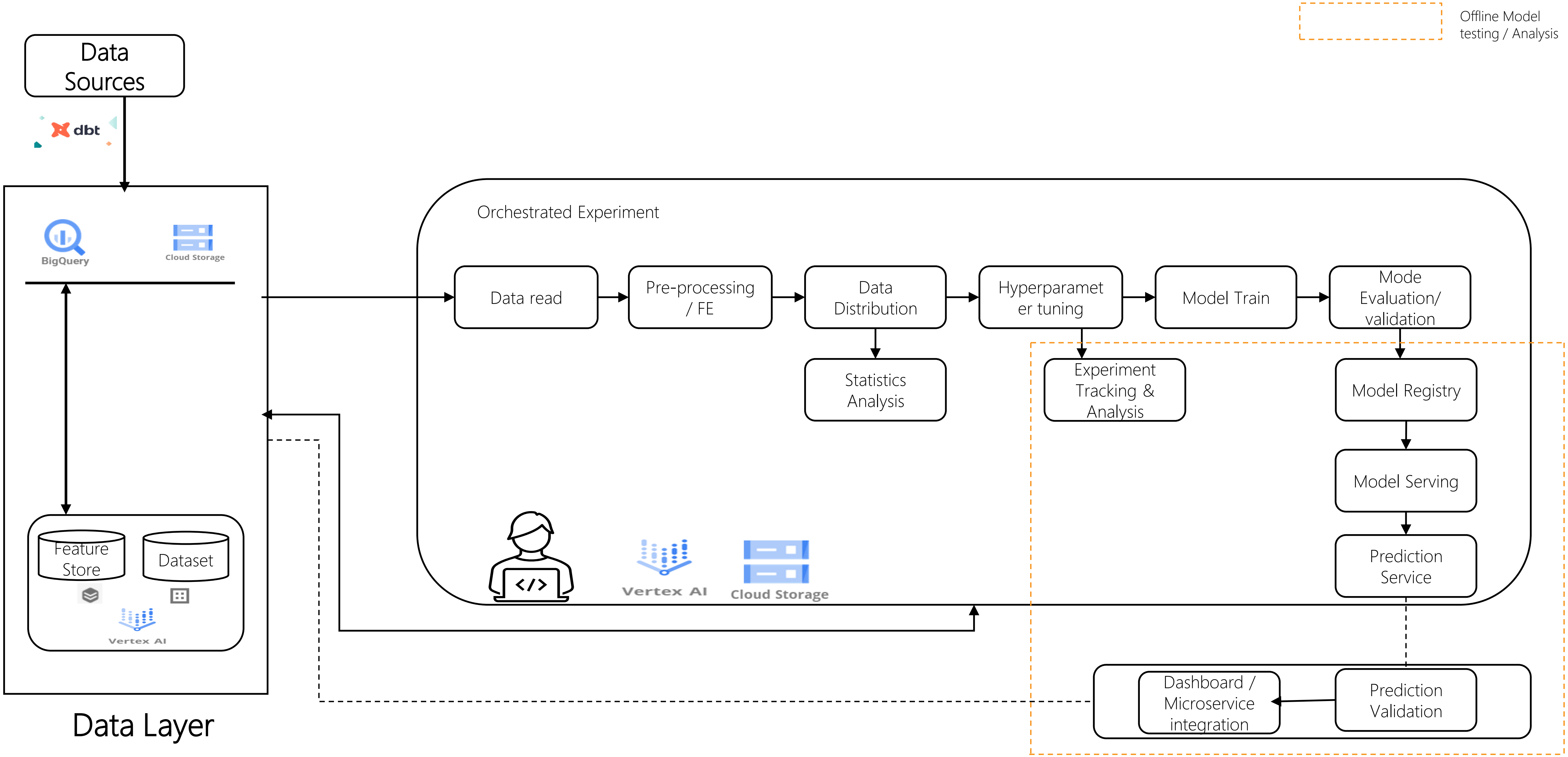
Data Science Experimentation Zone

Goal: Cloud based experimentation zone for Data Scientist for fast prototype and execution

- Framework features & benefits
  - Easy spin up of environment using templates
  - Resources like – vertex notebook, IDE, infrastructure such as bucket, cloud function, etc.
  - Access management made easy
  - Prototypes – with templates of various prototypes available, we can have faster experiments. Identify functions/use-case etc. and create a repo and templatize them. Some example could be data read / data validations , feature engineering components etc.
  - Within the GCP environment, data scientist can execute end-to-end pipeline testing as well and have the components ready for training/inferencing pipelines.

Priority: Low; GMI is using experimentation sandbox for their data science team and agreed to take this in future

Data Science Experimentation Zone



Training pipeline architecture

Goal: A standardized Model Training operationalization framework that provides robust observability capabilities of all system components

- Framework features & benefits
  - Largely automated & config-driven keeping customization and extensibility in mind
  - Generic, reusable component architecture for common data/training operations
  - Integrated metrics and monitoring functionality at project, pipeline, component, and package level that enables projects to have proper observability necessary to reliably support continuous model training
  - Incorporates DevOps best practices for testing and CICD

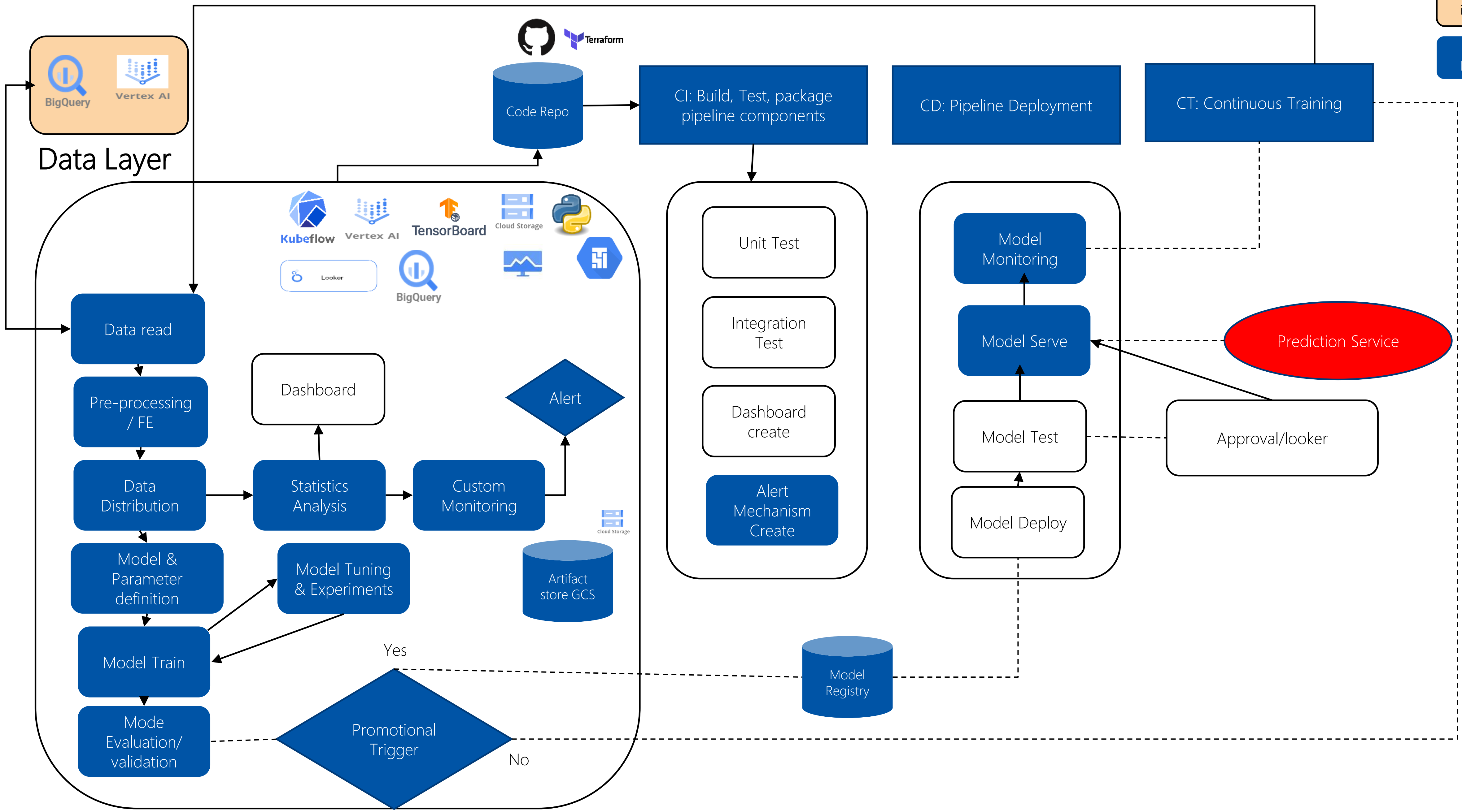
Automated model training pipeline architecture

Aspirational

Priority components

Central implementation

Project Implementation



BigQuery

Vertex AI

Data Layer

Terraform

Code Repo

CI: Build, Test, package pipeline components

CD: Pipeline Deployment

CT: Continuous Training

Kubeflow

Vertex AI

TensorBoard

Cloud Storage

Looker

BigQuery

Data read

Pre-processing / FE

Data Distribution

Statistics Analysis

Dashboard

Custom Monitoring

Alert

Model & Parameter definition

Model Tuning & Experiments

Cloud Storage

Artifact store GCS

Model Train

Yes

Mode Evaluation/ validation

Promotional Trigger

No

Unit Test

Integration Test

Dashboard create

Alert Mechanism Create

Model Monitoring

Model Serve

Model Test

Model Deploy

Prediction Service

Approval/looker

Model Registry

Model inferencing pipeline architecture

Goal: A standardized Model Inferencing operationalization approach that provides robust observability capabilities of all system components

- Framework features & benefits
  - Largely automated & config-driven keeping customization and extensibility in mind
  - Generic, reusable component architecture for common data/training operations
  - Integrated metrics and monitoring functionality with alert mechanism for various area of model, drift, service with dashboard
  - Incorporates DevOps best practices for testing and CICD

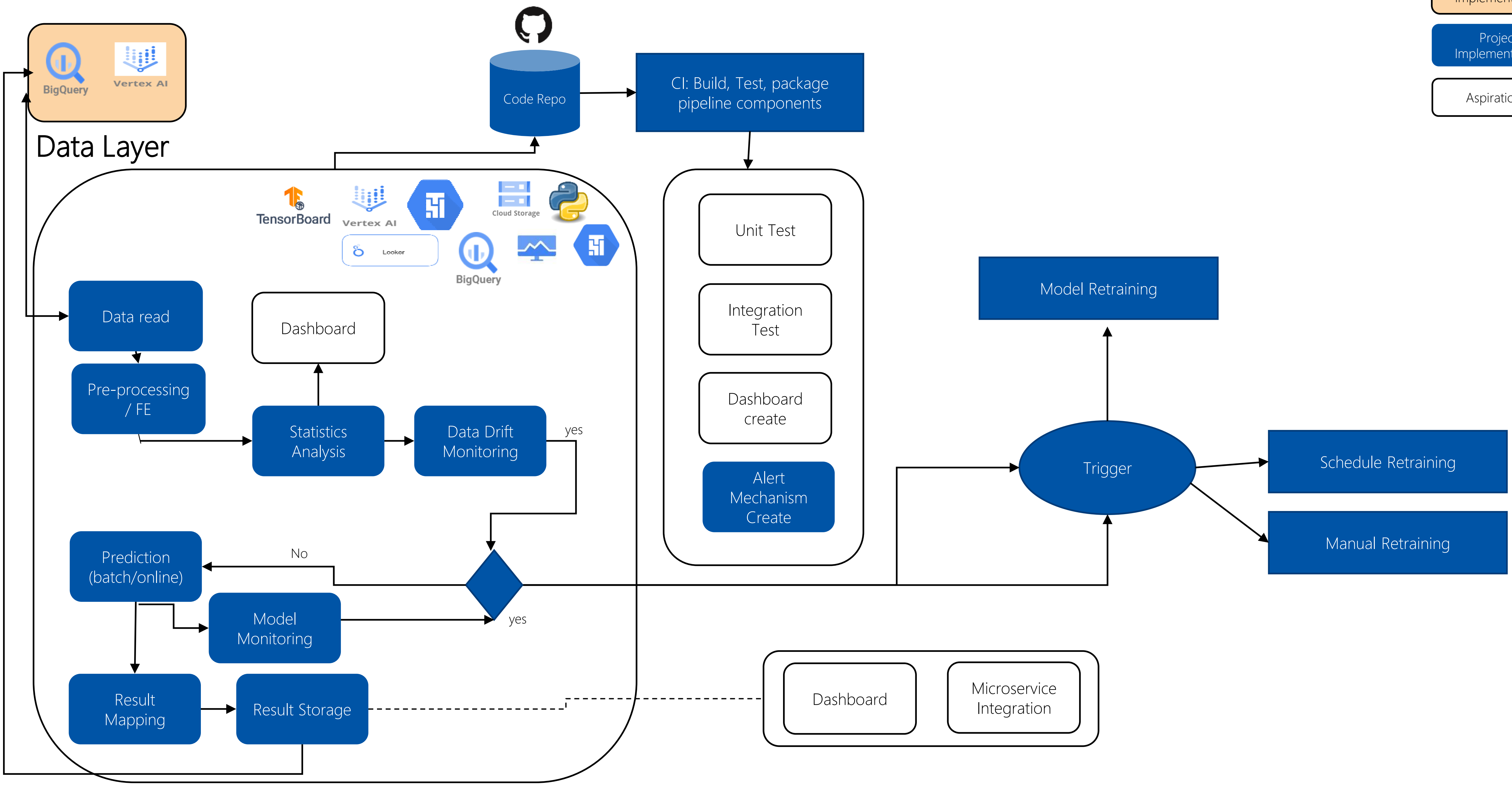
Automated inferencing pipeline architecture

Priority components

Central implementation

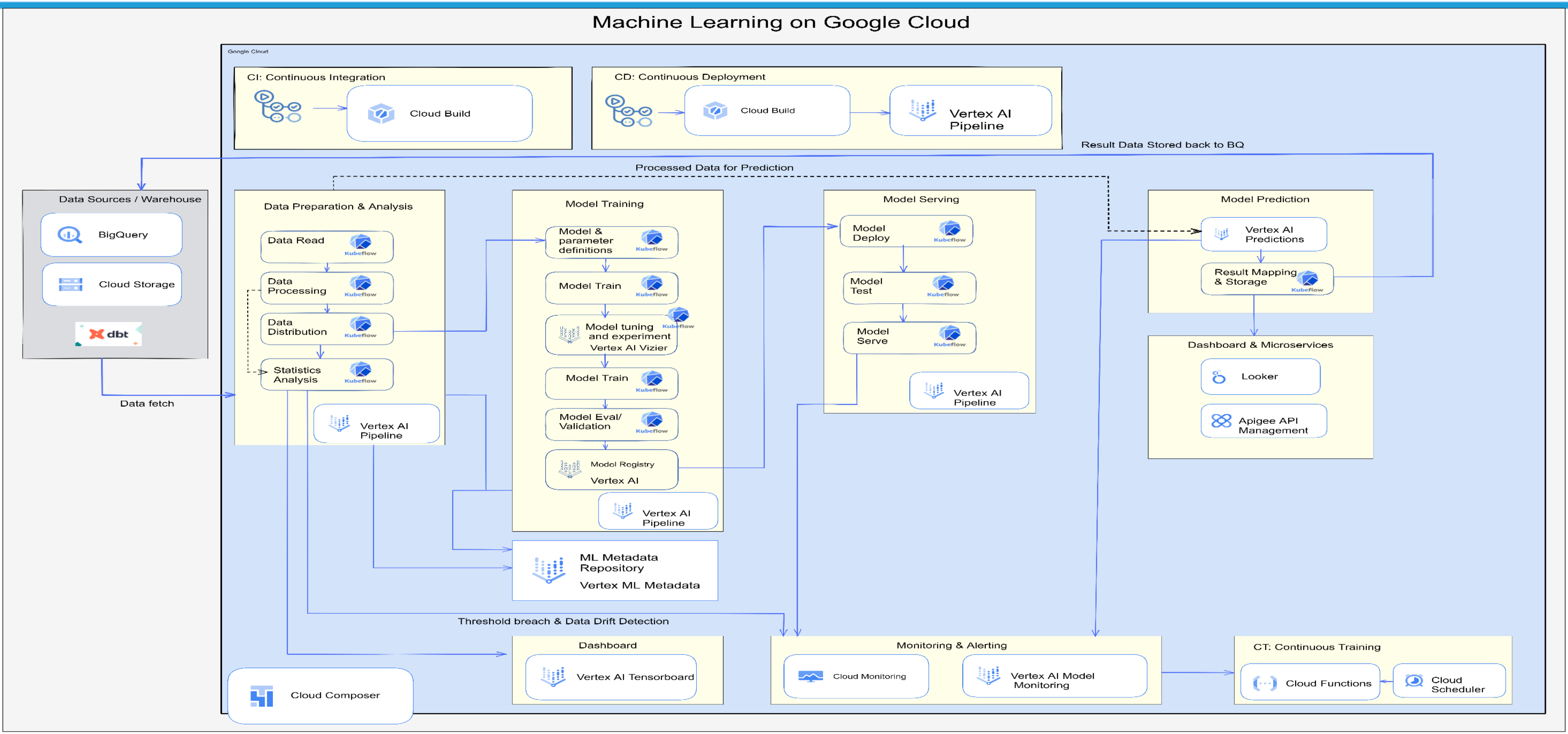
Project Implementation

Aspirational



Solution design & tools

Data and Analytics



Machine Learning on Google Cloud

Google Cloud

CI: Continuous Integration

Cloud Build

CD: Continuous Deployment

Cloud Build

Vertex AI Pipeline

Result Data Stored back to BQ

Processed Data for Prediction

Data Sources / Warehouse

BigQuery

Cloud Storage

dbt

Data fetch

Data Preparation & Analysis

Data Read

Data Processing

Data Distribution

Statistics Analysis

Vertex AI Pipeline

Model Training

Model & parameter definitions

Model Train

Model tuning and experiment Vertex AI Vizier

Model Train

Model Eval/ Validation

Model Registry Vertex AI

Vertex AI Pipeline

Model Serving

Model Deploy

Model Test

Model Serve

Vertex AI Pipeline

Model Prediction

Vertex AI Predictions

Result Mapping & Storage

Dashboard & Microservices

Looker

Apigee API Management

ML Metadata Repository

Vertex ML Metadata

Threshold breach & Data Drift Detection

Dashboard

Vertex AI Tensorboard

Monitoring & Alerting

Cloud Monitoring

Vertex AI Model Monitoring

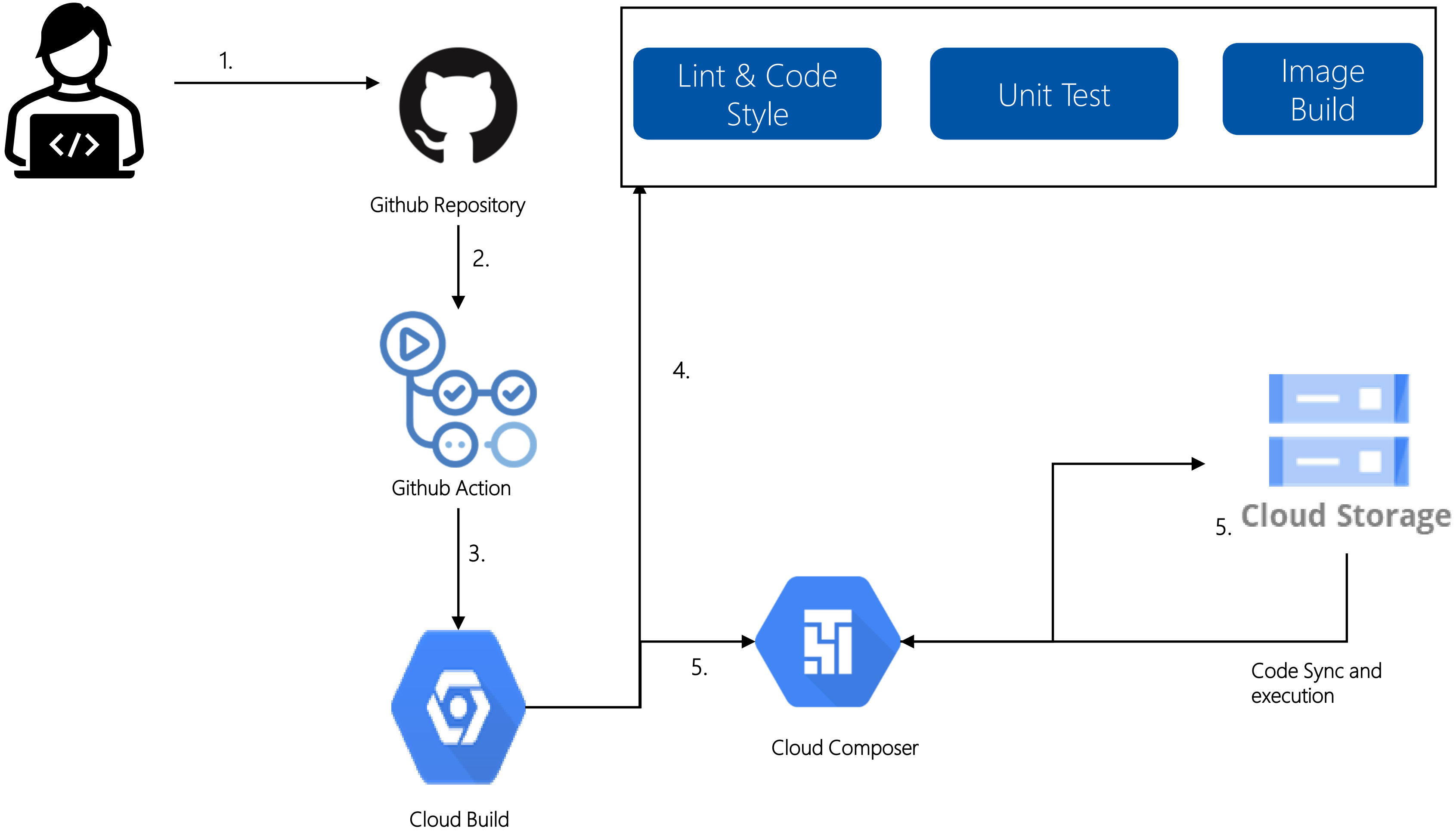
CT: Continuous Training

Cloud Functions

Cloud Scheduler

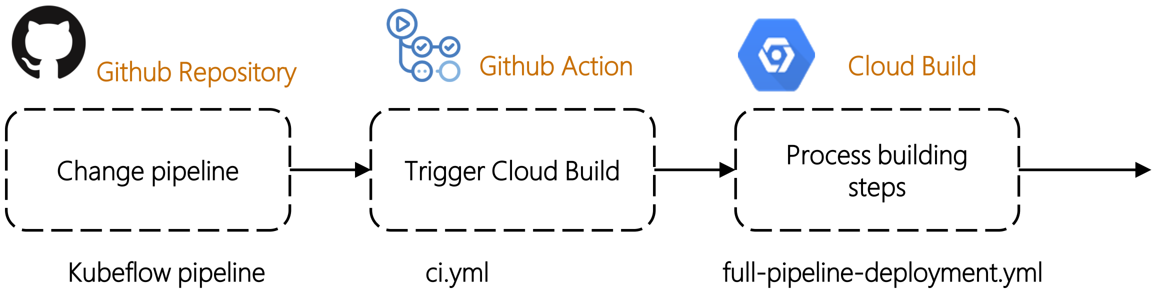
Cloud Composer

CI/CD approach explained (overall)



1. Developer pushes code to the repository
2. Git Actions triggers which has workflows. Workflow activation
3. Cloud build triggers and starts with running the workflow
4. Unit test and other test triggers, once passes will build images and push them
5. code sync with composer

CI/CD explained (pipeline)



Github Repository

Change pipeline

Kubeflow pipeline

Github Action

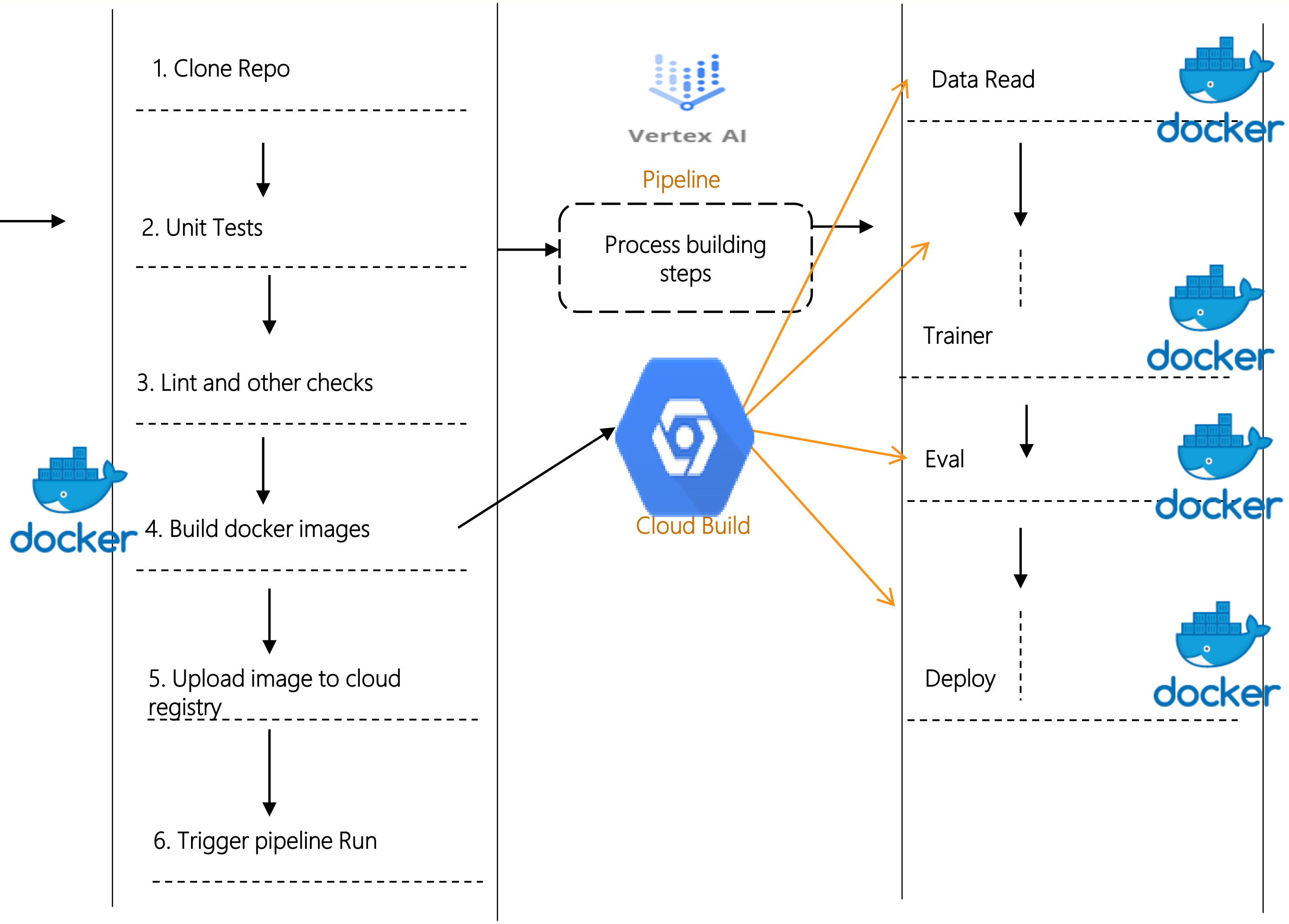
Trigger Cloud Build

ci.yml

Cloud Build

Process building steps

full-pipeline-deployment.yml



1. Clone Repo

2. Unit Tests

3. Lint and other checks

4. Build docker images

5. Upload image to cloud registry

6. Trigger pipeline Run

Vertex AI

Pipeline

Process building steps

Cloud Build

Data Read

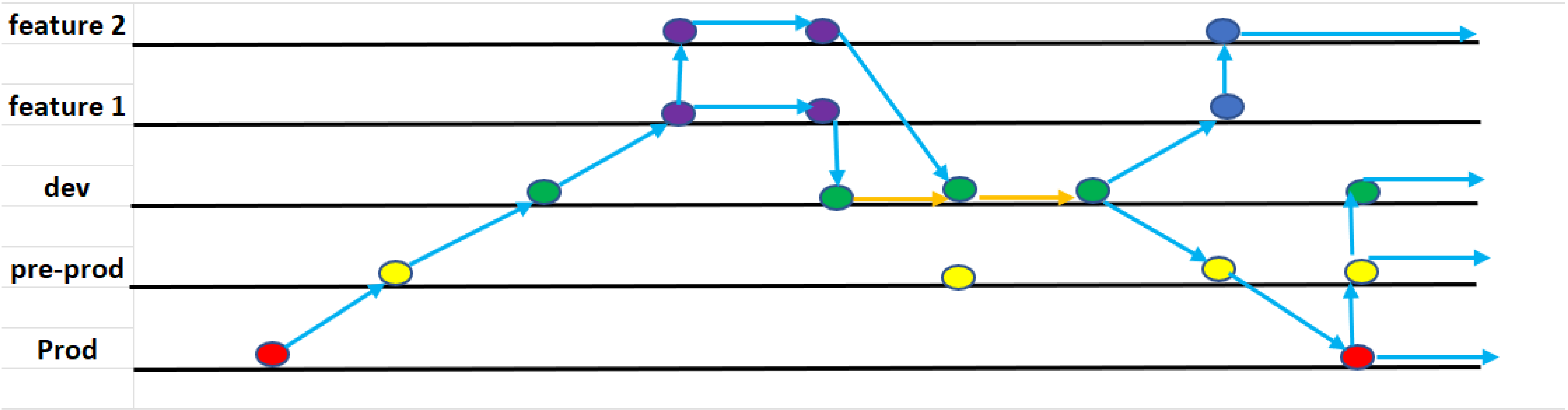
Trainer

Eval

Deploy

1. Developer pushes pipeline code to the repository
2. Git Actions triggers which has workflows. Workflow activation
3. Cloud build triggers and starts with processing the workflow steps
4. After integration, component images gets build and pushed to container registry
5. Each images getting called while executing components on vertex

Branching strategy



🔴 Main master branch, production ready code

🟡 Pre-prod, before releasing to main/master

🟢 Up-to-date Dev branch, with latest developed features

🔵 Feature branch for each developer to work independently on new functionality

- Code migration will occur via gitaction yaml files post pull request by developer
- Each yaml file will call actions within and execute the integration steps

* - We will utilize GMI repo structure for our development phase

Appendix