Pilot Platform Technical Whitepaper

Version 1.0: November 23, 2022

Abstract

Pilot is a comprehensive data platform solution developed by Indoc Research for secure management, analysis and sharing of medical research data. Here, we describe the design and architecture of the Pilot data platform, as well as the requirements and rationale behind the technical solutions. This whitepaper is a living document and will be updated as Pilot evolves to meet the needs of different research programs.

1 Introduction

The secure management and responsible sharing of digital health data is a major challenge faced by hospitals, academic institutions, governments, and industry partners. There is an unmet demand for technical solutions to facilitate the management of sensitive and diverse types of health data in a secure manner while being compliant with data privacy or protection laws such as Personal Health Information Protection Act (PHIPA) in Ontario, Canada, Health Insurance Portability and Accountability Act (HIPAA) in the United States, and General Data Protection Regulation (GDPR) in the European Union (EU). In addition, the solutions need to satisfy internal security practices and polices set by the organizations managing the data.

While protecting data, a platform for managing health data must meet additional requirements. The data platform must be capable of securely integrating with various services and applications, including electronic health records, clinical data management systems, imaging, and laboratory systems, as well as IT solutions such as virtualization, network, storage, identity, and security infrastructure. The platform also needs to be flexible in ingesting and annotating a wide range of data types such as clinical data elements, lifestyle surveys, radiological and pathology images, genetic tests and genomic sequencing, and wearable data. Such a data platform must also be scalable with increasing volume and complexity of data, and growing number of users in concurrently running projects.

Pilot is an open-source solution for secure management, analysis, and sharing of medical research data. It is developed by Indoc Research, a not-for-profit company dedicated to the design, build, and maintenance of data platforms for hospitals, universities, research institutes, government, and large-scale research programs. Pilot can be deployed in a cloud environment, on-premises infrastructure, or in a hybrid configuration allowing the optimized utilization and flexible expansion of computing resources. Multiple instances of Pilot can be deployed in different geographical locations or jurisdictions and interoperate to support data federation in compliance with data residency requirements.

Pilot is powering several data platforms for large-scale research programs and is in active development to support a wide range of use cases. The Virtual Research Environment (VRE, https://vre.charite.de/) is a customized instance of Pilot deployed at the Berlin Institute of Health at Charité (BIH). The security and privacy measures of the VRE have been independently validated to be in adherence to GDPR requirements for the processing of personal health data for research purposes. The Health Data Cloud (HDC, https://www.healthdatacloud.eu/) is being developed by Indoc in partnership with the EBRAINS

neuroscience research data platform stemming from the Human Brain Project, together with Charité and various academic and high performance computing centres to deploy distributed data management services for the management and sharing of sensitive data across EU and non-EU countries. Building upon the HDC, the eBRAIN-Health project funded by the European Commission aims to provide a GDPR-compliant platform for collating complex neurobiological data to simulate "digital brain twins" for innovative research. Brain-CODE (https://www.braincode.ca) is another specialized instance of Pilot supporting the Ontario Brain Institute's large-scale research programs, as is the Ontario Health Data Platform (OHDP, https://ohdp.ca), a secure infrastructure containing health system and administrative data from 15 million Ontarians as well as genomic variant and COVID outcomes, among others.

2 Design of Pilot

2.1 Design Principles

The design, development and operation of the Pilot platform is guided by several principles.

FAIR Principles. Pilot follows and promotes the foundational principles of making data Findable, Accessible, Interoperable, and Reusable[1] (FAIR). Pilot supports annotation of research data with rich metadata, adopts ontologies and standardized schemas where applicable, and enables the data to be searchable and retrievable for downstream research.

Data protection by design and by default [2]***.*** The Pilot platform is designed with appropriate technical and organizational measures as part of its development and operations. Privacy and security features are built into the system and enabled by default. All data on the platform can only be accessed and used in the manner for which it was meant and by whom it was meant for. As a multi-tenant platform, isolation between projects is strictly enforced. Even within a project, there are multiple data zones of isolation so that only members with the right permission can access non-pseudonymized or protected data.

Scalability. Pilot can support additional concurrent users and operate on growing quantities of data by scaling horizontally onto additional compute resources. This will ensure consistent performance for all users even when there is a burst of compute intensive activities. For example, large concurrent uploads occurring on the platform will minimally affect other users exploring and analyzing data on the platform.

Extensibility. The Pilot platform can integrate or federate with third party software as needed by users. Such software applications include data capture tools, domain-specific analysis and visualization tools, and bespoke artificial intelligence solutions. Any candidate software is evaluated to ensure it can interoperate securely with Pilot and complies with the platform's security model.

Ecosystem driven. As we grow an ecosystem of research data platforms powered by Pilot, knowledge, policies, and technological advances developed for one program are available to the broader research community. As an open-source software, features developed for one research program will be available to other programs through the general releases of Pilot.

2.2 Zoning Structure

The Pilot platform implements a zoning structure. Each zone, defined by the Kubernetes namespace, can include its dedicated computing resources, storage locations, and platform services. The data traffic and

service communication between the zones are managed to prevent unauthorized data access. While the number of zones and their purposes can be customized based on needs, there are by default three primary zones: Green Room, Core Zone, and Utility Zone.

The ***Green Room*** is a data landing zone for any incoming data into the Platform. It is a designated environment with highly restricted access control suitable for highly sensitive data such as hospital data from electronic health records (EHRs), picture archiving and communication systems (PACS), laboratories, and other sources that may contain highly sensitive information such as direct identifiers. Inside the Green Room, the sensitive data can be pseudonymized, transformed, and prepared prior to being transferred to the Core Zone for broader access and general research use. The Green Room is deployed for each project on the platform and is isolated from other zones in terms of data storage and data processing functions to restrict access to only authorized users.

The ***Core Zone*** After the completion of pseudonymization or other preprocessing steps, Project Administrators can approve the data to be copied from the Green Room into the Core Zone. The Core Zone allows pseudonymized data to be securely stored, shared, and analyzed by all Project Collaborators.

The ***Utility Zone*** hosts the platform services and infrastructure components such as ingress, identity management, operational databases, Application Programming Interface (API) routing, and notifications. No user data or research data is stored in the Utility Zone.

This zoning structure can be useful in several scenarios. It helps to isolate data of different sensitivities providing different groups of users with appropriate access as the data moves along its lifecycle. For instance, the Green Room serves as the initial data landing zone where access must be restricted. Only after the pseudonymization process will the data be approved to move into the Core Zone where broader user access is allowed.

Another use case of the zoning structure is for grouping the platform services by similar functionalities or operation needs. Here are some examples: the platform may include a zone for object storage cluster to allow the compute and storage to scale independently; a workspace zone is created for each research project on the Pilot providing researchers an interactive analytic environment; a highly restricted Secret Vault zone can be created to store operational secrets; there can be dedicated zones for knowledge graph and pipeline registry.

3 Overview of the Pilot Platform

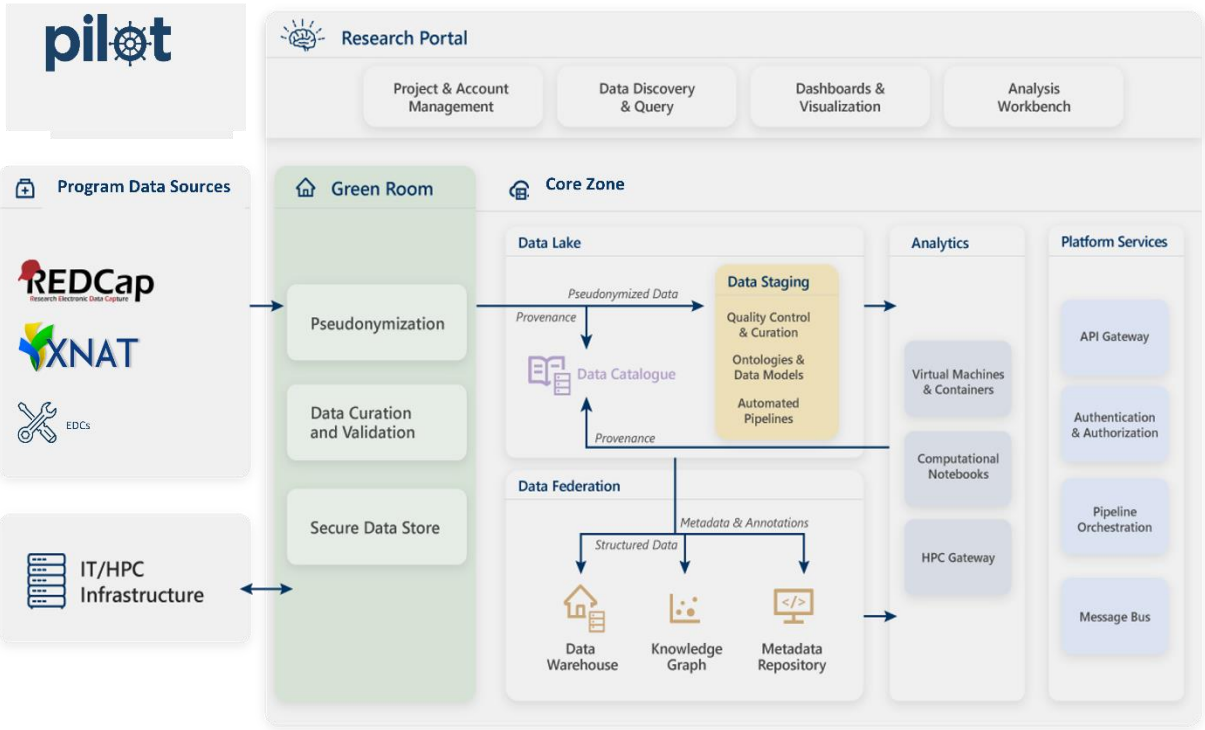


Figure 1. The Pilot Data Platform

A high-level architecture of the Pilot platform is shown in Fig. 1. Users interact with the Pilot platform primarily through the Research Portal, a web application for accessing and managing resources and data on the platform. Data in Pilot is isolated into Projects with data storage and processing containers that hold all the data files for a research project. A Platform Administrator creates each Project and assigns a Project Administrator to control user management for project members. Through project-based and role-based access control, user access to data and resources can be fine tuned based on governance and policies established by the platform sponsor. The Pilot platform also includes a federated identity management system and segregated data zones tailored for processing data with different access control requirements depending on the level of data sensitivity.

Pilot supports a wide range of data types and facilitates data ingestion from different electronic data capture applications such as REDCap for electronic health records (EHRs), XNAT for radiological images and Laboratory Information Management Systems (LIMS) for genomic sequencing data. The segregated Green Room is a landing zone for incoming data and can be configured with tools or automated pipelines to pseudonymize and pre-process sensitive data such as EHRs. The processed data can then be made available in the Core Zone for broader research use.

Each project is provided with analytics workspaces for processing, analyzing and visualizing datasets. The workspaces can be configured with JupyterHub for coding notebooks in Python or R, Apache Superset as business intelligence tools, remote desktop access to virtual machines, and integration with high performance computing infrastructure. In addition, there are project-specific data warehouses for

structured datasets, a metadata repository and graph database for querying metadata and tracking data lineages. The platform also provides support for ontologies and metadata schemas to enable discoverability of data.

4 Platform Architecture

The Pilot platform is based on a layered microservices architecture where the responsibilities within each layer are further organized into components (Fig. 2). The business logic that drives the features of the platform resides in a microservices layer. Following the principles of Domain-Driven Design (DDD), the platform is composed of small services that can be developed, managed, and scaled independently of one another. The management of the services are further simplified through containerization and orchestration by Kubernetes. The persistence layers handle all data on the platform utilizing different storage mechanisms. Asynchronous tasks on the platform are managed through queues and events.

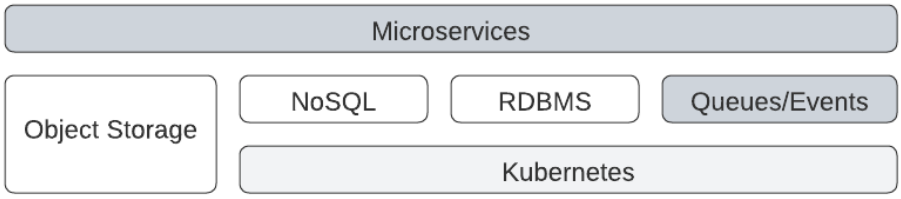


Figure 2. A Layered microservices architecture of the Pilot platform

4.1 Microservices Layer

The components in the microservices layer are described briefly here, and the interaction between these microservices are illustrated in Fig. 3.

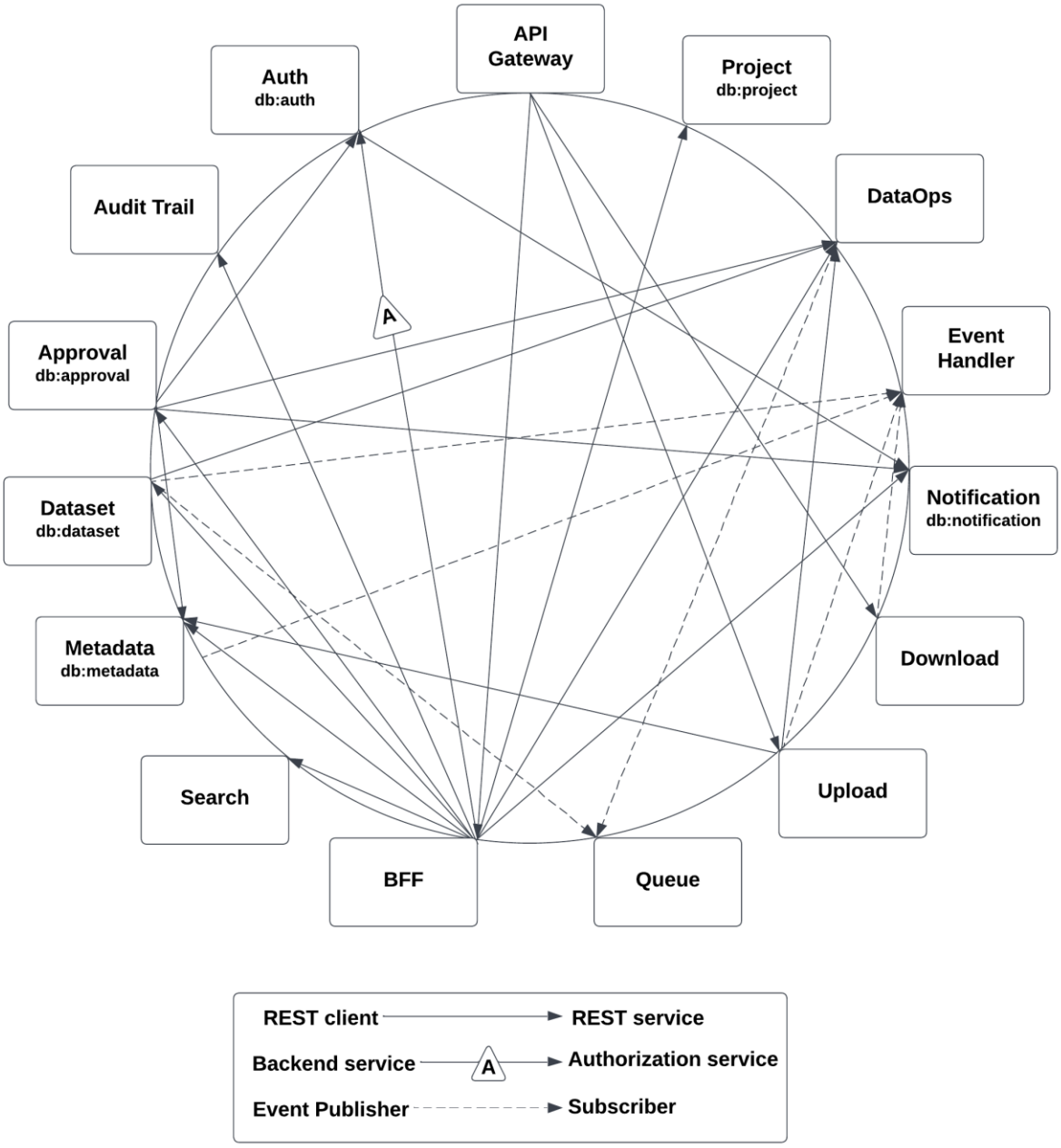


Figure 3. Microservices supporting the Pilot Platform and their interactions.

API Gateway. API Gateway is a single-entry point situated between the Ingress Controller and the platform services. The API Gateway is responsible for request routing, composition, rate limiting, and monitoring. It handles requests by routing them to the appropriate back-end services. If there are failures in the back-end services, the API Gateway can return cached or default data. The API Gateway is also connected to the internal Identity Provider (IdP) to protect the back-end APIs from unauthorized access.

Auth. The Auth service provides user authentication and authorization by integrating with Keycloak, an open-source solution for identity and access management (IAM). Keycloak serves as an Identity Provider

(IdP) issuing identities to users and enables Pilot to federate institutional identities in Microsoft Active Directory or Lightweight Directory Access Protocol (LDAP) system at organizations where Pilot is deployed.

This Auth service is responsible for creating, modifying, and deactivating user accounts. The service utilizes a role-based policy to control all API endpoint access. User projects, groups, and roles configured in Keycloak are used to manage access to the platform components and data. Authorization to the platform is performed through OAuth 2.0 protocol allowing the platform to not only integrate with third party tools, but also supporting single sign on (SSO) across these tools.

Audit Trail. Audit Trail service generates and maintains the data lineage that gives users the visibility of the data life cycle, including the complete data flow from its source to consumption as well as the metadata associated at each stage.

Project. Project service is responsible for creating and modifying a project on the Pilot platform. It also provides a set of backend APIs to provision and manage workbench resources which are dedicated applications provided to each project for analysis and visualization.

DataOps. DataOps service performs general data operations on file objects, including annotating, tagging, and organizing files into collections. By default, there are two versions of DataOps service, one is deployed in the Green Room and the other in the Core Zone. Each version can only access the data residing in the same data zone.

***Metadata*.** Metadata service is a major component of the Pilot platform, managing the metadata stored and warehoused in the system. It is responsible for creating and managing the metadata which includes owner, storage location, tag, manifest and other data related information. The metadata service also provides several approaches for data controllers to better organize their data, such as creating collections or favouriting items.

Dataset. Dataset service is one of the major components on the Pilot platform and provides dataset metadata management capability. It supports real-time tracking on the dataset activities and allows users to generate a snapshot of a dataset at any given time. The generated snapshot includes not only data within dataset but also the metadata used to describe the dataset.

Search. Search service is responsible for providing rich query and aggregation functionalities. It is backed by Elasticsearch, an open-source search engine and document index that stores data in JSON format. Search service uses Elasticsearch to provide scalable full-text search for metadata and activity logs. In addition, it provides a set of APIs to aggregate the project activities in nearly real-time.

Upload. Upload service supports file uploading by generating a temporary presigned URL that allows user to connect directly with the object storage to upload data. For large file uploading, the upload service uses a 'multipart upload' approach to transfer data in small chunks and combine them when all parts land in the backend storage.

Download. This service receives user download request, bundles requested files into a zip package, and generates a temporary accessible download URL for users to download the file from the Pilot platform.

Queue. Queue service as a message queuing service eliminates the complexity and overhead associated with managing RabbitMQ, an open-source message-broker software. Queue service provides a simple API

endpoint to upstream services for sending messages to RabbitMQ and consuming messages received in a queue.

Pipelinewatch. Pipelinewatch is connected with the Kubernetes cluster for monitoring job execution status. It integrates with Kubernetes watch to fetch the event stream and save the consequential job status to Redis for future querying.

Event Handler. Event Handler connects with Kafka, an open-source event streaming platform, to receive all stream events, and stores them in Elasticsearch after proper schema verification.

Notification. Notification service is responsible for sending outgoing notifications to users such as email and announcements. It also provides the functionality to manage platform-wide maintenance notifications that are displayed in the user interface of the platform.

Approval. Approval service is responsible for managing user copy requests. It allows users with appropriate permission to submit and approve data copy requests. The service provides a secure way to curate and share data with project members by supporting physical data movement between different data zones.

BFF-Web (Backend for Frontend - Web). BFF-Web is a service dispatcher that manages the routing of user requests received from the web user interface. It also aggregates and optimizes the results returned from the backend services before returning the response to browser clients.

BFF-CLI (Backend for Frontend - Command Line Tool). Similar to BFF-Web, BFF-CLI is consumed by the Command Line Interface instead of the web portal.

4.2 Kubernetes Infrastructure

To ensure the Pilot Platform is portable to deploy in a variety of environments, the individual platform services are distributed as containerized software with all their runtime software dependencies packaged in the container. To manage these containers and their network and storage dependencies, we make use of a container orchestration platform, Kubernetes. This technology allows powerful management of containerized applications with robust security and monitoring features. Importantly, Kubernetes is the software industry standard for container management, and as such, has great support and compatibility with a large variety of environments. Thus, by targeting Kubernetes, which can be deployed anywhere, we can offer the Pilot Platform as a stable deployment target.

4.3 Persistence Layer

The persistence layer is responsible for storing and retrieving application data. Pilot utilizes three types of storage mechanisms in this layer.

Relational Data. For data that describes the various entities, their metadata, and their relations to one another, the Pilot Platform makes use of PostgreSQL as its Relational Database Management System (RDBMS). PostgreSQL is a popular open-source solution for relational data, providing exceptional performance and advanced SQL features for modelling and querying data.

Non-relational Data. The Pilot Platform makes use of Non-relational and NoSQL data stores. For facilitating fast free text search and aggregations the platform leverages Elasticsearch. For key-value data used for caching, the platform makes use of Redis.

Object Storage. For handling raw data such as imaging, genomic, text, and models, the platform makes use of object storage. Object storage is well suited for handling the secure and encrypted storage of files as it is designed for scale. The Pilot Platform currently targets the S3 Object Storage API created by Amazon and uses MinIO to provide these capabilities allowing it to be deployed anywhere.

4.4 Message Queues and Event Bus

Longer running asynchronous tasks can be managed with message queues allowing users and services to create jobs in the backend. RabbitMQ is a message broker that allows services within the platform architecture to communicate asynchronously with each other. For streaming event data for auditing and search purposes, the platform makes use of Kafka which is an event streaming platform.

5 Platform Deployment and Delivery

The successful operation of a data platform requires robust and efficient deployment processes as well as timely and consistent delivery of updates, fixes, and feature additions. Importantly, the processes and technologies of choice need to ensure that the stability of the platform is guarded. This section describes the tools and processes for managing, building, and publishing changes in the Pilot source code, as well as deploying new versions of the software to platforms already in production.

5.1 Git Repository

Every software component, be it a web service, event handler, CLI tool, or software library follows similar development flow and practices. Each component lives in a git repository where the code resides, changes to the code are managed, and it is the source of truth for versioning of the software.

Important files in the git repository can include:

- Project and dependency definition files
  - `pyproject.toml` - used by the poetry build tool for Python based projects. Contains version of the code base, dependencies, and build configuration.
  - `package.json` - used by npm for our Javascript based projects. Manages version, dependencies, and build scripts.
- `Dockerfile` - defines the Open Container Initiative (OCI) compatible images that describe how our containerized software is built and run
- `.github/workflows/*` - pipeline definition for testing, tagging, and building the software using GitHub actions

The evolution of the code base pertaining to feature development, refactoring, and bug fixes follows a Git Flow style workflow. There are two key branches: `main`, which is the latest stable release of the software, and `develop`, which is the latest edge version of the software under active development. Bug fixes and features are branched off the develop branch and new releases are promoted up to the main branch

where they are tagged and published. Any hotfixes made to released software is merged back to the develop branch. Fig.4 below illustrates the workflow.

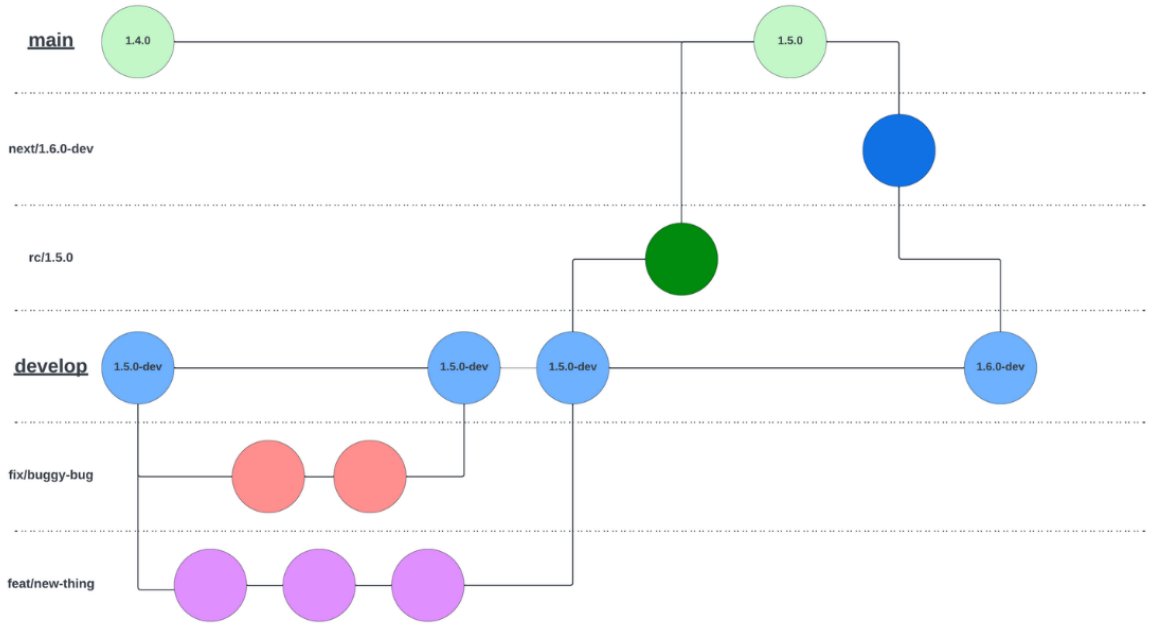


Figure 4. The Git workflow that involves the use of feature, RC, and other branches, keeping main branch clean and permitting concurrent development on the same feature

5.2 Container Registry

Most of the services are built and published as docker images. These images are published to a container registry so that they are publicly available. In the case of Pilot, these are published to the GitHub container registry (ghcr).

5.3 Infrastructure Repository

The first step to initiate a tangible deployment of the Pilot platform is to set up the git repository that hosts all infrastructure-as-code (IaC), configuration, and automation. This repository is used to bring up and manage the infrastructure as well as deploying services and updates made by developers.

The setup and infra directories illustrated in Fig. 5 are used to create the infrastructure. Any onetime setup and bootstrapping scripts that create service accounts and initial resources will live in the setup directory while the infra directory will contain the infrastructure-as-code in the form of Terraform that describes and evolves with the infrastructure needs.

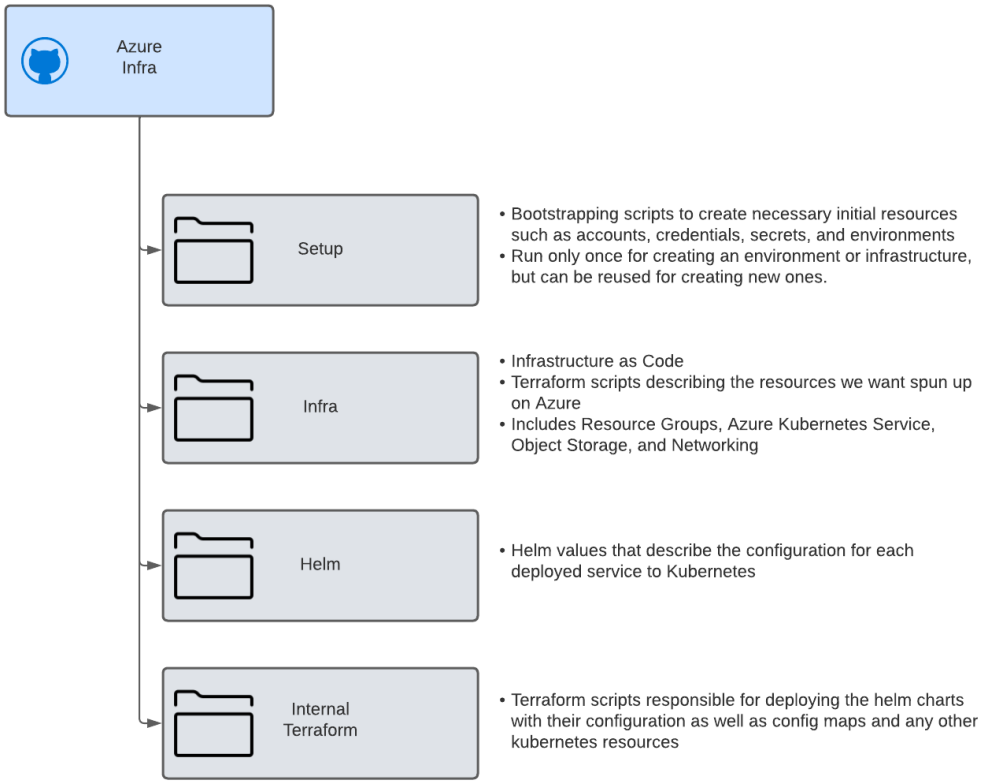


Figure 5. Directories setup for creating the Pilot infrastructure

5.4 Helm Charts

Helm charts are templates for creating a deployment (or other resource types) within Kubernetes. Charts can be opinionated and have control over what values they accept and provide to the underlying template. The values file within a published chart will contain safe default values. A very important feature provided by Helm is the ability to perform rolling updates of services, record deployment history, and easily roll back deployments. This allows for zero or near-zero downtime updates to services while minimizing impact to active users of the platform.

Helm Chart Repository. This repository hosts the Helm charts for our services which are templates for creating a deployment (or other resource types) within Kubernetes. Charts can be opinionated and have control over what values they accept and provide to the underlying template. The values file within a published chart will contain safe default values.

Helm Values. A directory structure holds the Helm values for each service as it relates to each environment. These values are part of the infrastructure repository described above.

```
$ tree helm
helm
├── mailhog
│   └── dev
│       └── values.yaml
└── service_metadata
│   └── dev
│       └── values.yaml
└── service_dataset
    └── dev
        └── values.yaml
```

Any changes to the configuration of a service can be made here. For example, the HPC service has a value file that looks like this:

```
image:
  repository: hpc
  tag: 25
  pullPolicy: Always

container:
  port: 5080

service:
  type: LoadBalancer
  port: 5080
  targetPort: 5080

nodeSelector:
  namespace: utility

extraEnv:
  port: 5080
  log_level: info
```

5.5 Terraform

Terraform is the tool describing what charts are deployed, where they are deployed, with what versions and values. Terraform takes care of updating the infrastructure, reporting any differences, and managing the state and lifecycle of our deployments. This Terraform resides in the infrastructure repo.

Following the example from above where we have the HPC service described as a Helm chart with values ready for it in dev. To deploy it, our Terraform would have:

```
resource "helm_release" "hpc" {

  name = "hpc-example"

  repository = "https://pilotdataplatform.github.io/helm-charts/"
  chart      = "hpc-service"
  version    = var.hpc_chart_version
  namespace  = "utility"

  values = [file("../helm/service_hpc/${var.env}/values.yaml")]

  set {
    name  = "image.tag"
    value = var.hpc_app_version
  }
}
```

The version of the chart and application for that environment are described in a variables file specific to that environment:

```
env       = "dev"

# HPC Service
hpc_chart_version = "0.1.0"
hpc_app_version   = "25"
```

5.6 CI/CD

CI/CD is the combination of continuous integration and continuous delivery. Pilot makes use of GitHub actions to test, build, and publish new versions of the software as well as an open-source solution called Jenkins for automating the deployment of services.

5.7 End-to-End Workflow

The tools and processes described above provide great separation of different concerns, removing deployment specific logic from the application code base while maintaining a description of our infrastructure-as-code (IaC) in a git repo. These different processes can be stitched together with automation. The high level workflow is illustrated in Fig. 6.

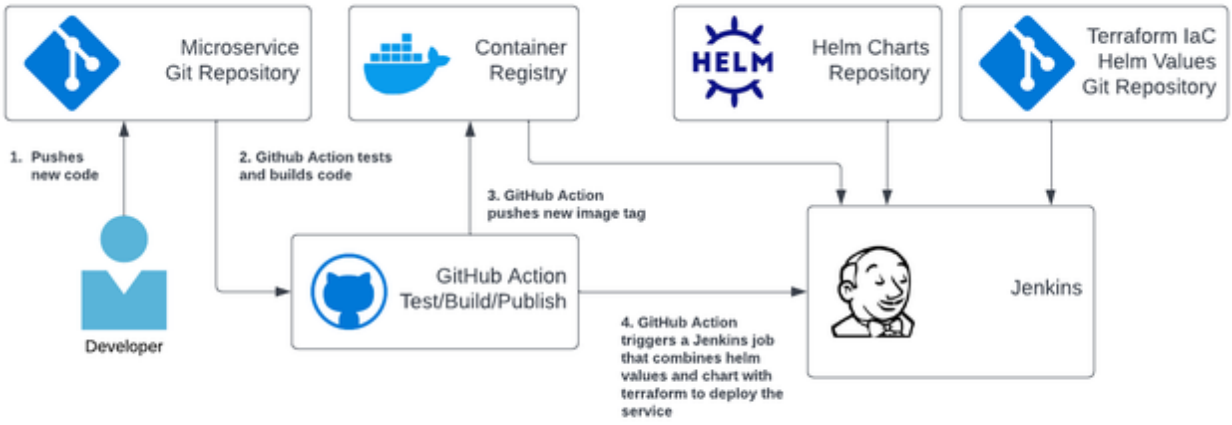
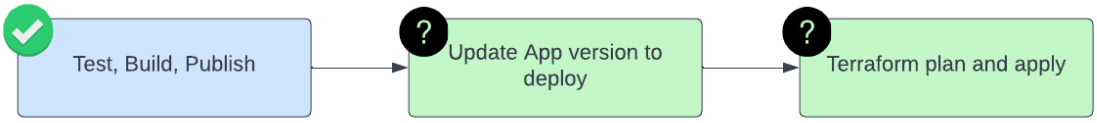


Figure 6. Workflow for Pilot deployment and delivery.

Steps 1, 2, and 3 in this workflow are self explanatory, but step 4 has some internal details that require explanation.



Updating Service Versions. If we use the same example service as before:

```
# HPC Service
hpc_chart_version = "0.1.0"
hpc_app_version   = "25"
```

We want to be able to update the `app_version` variable. This can be done manually by a human in the scenario where we are releasing a new version to some production environment. But what about maintaining our CI/CD practices in a dev environment? This can be done through Jenkins with a simple pipeline that gets triggered on the completion of a develop/main build.

```
def targetEnv = env.TARGET_ENV
def targetRelease = env.TARGET_RELEASE
def newAppVersion = env.NEW_APP_VERSION
def causes = currentBuild.getBuildCauses()

def updateCommand = """sed -r -i -e \\
    "s/(^${targetRelease}_app_version\\s*=\\s*).*/\\1\\"${newAppVersion}\\"/"
\\
    config/${targetEnv}/${targetEnv}.tfvars
"""
```

Successful jobs from a develop branch can trigger this Jenkins pipeline to update the app version in the git repo which in turn triggers the Jenkins pipeline that applies the Terraform.

Applying Terraform. Running the Terraform that describes our architecture occurs in three steps.

1. `terraform init` - This pulls down the latest recorded state of the architecture from the configured backend. Terraform uses both the last recorded state from the last run and what it actually sees deployed to help determine the actions it needs to take.
2. `terraform plan` - Terraform runs and computes the differences and actions that it needs to take. It will output the result of this to a plan file that it will later apply. By using a plan file, we can review the changes and modify if necessary, but it also provides a deterministic way of applying changes as the system could be changing in real time and we want to ensure what we plan is what is applied.
3. `terraform apply` - Applies the previously computed plan. Terraform will notify you of all successes, failures, and warning, and record those in its state that it stores in its configured backend.

The figure below (Fig. 7) demonstrates the end-to-end flow from changes being merged into a codebase to building and deploying that code as viewed in GitHub Actions and Jenkins.

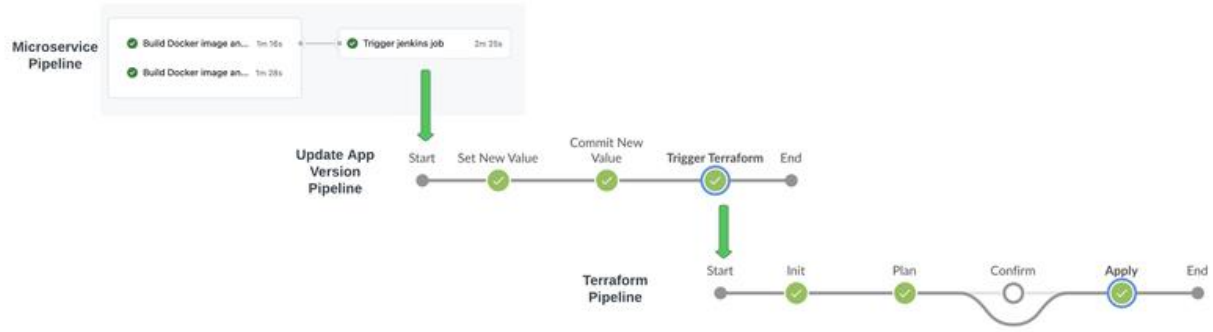


Figure 7. End-to-end workflow from merging code changes to building and deploying of the code.

6 Source Code and Technical Documentation

The Pilot source code and API documentation are available at https://github.com/pilotDataPlatform. Helm charts as described above are also available for deploying Pilot. Following an agile software development methodology, new versions of the Pilot software components will be regularly released to Github. As an open-source initiative, we welcome code contributions, feature requests and issue reporting from researchers and developers.

7 References

1. Wilkinson MD *et al*. The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* (2016) Mar 15; 3:160018.
2. General Data Protection Regulation (GDPR) Article 25