THETARAY

Version 6.13

Financial Crime Compliance Suite

TECHNICAL OVERVIEW

Document

Revision A

Publication Date: April, 2025

Legal Notice:

1. Preface

ThetaRay 6.0 system is a distributed system deployed into an Openshift / Kubernetes environment that manages the full lifecycle of solutions used to detect and analyze risky financial activities. The system collects data, processes it, uses full stack methodologies to detect risks and provides an analyst/ data scientist facing application for interactively analyzing alerts.

This document presents a high level system architecture, the various component sub-systems and their roles and the compute / storage requirements needed for deploying the system.

This document is primarily aimed at customers' data scientists and operations personnel responsible for the integration of ThetaRay Analytics Platform into their environment(s).

The document covers the following main topics:

- System Overview
- Architecture Overview
- Security
- Deployment
- Resource Requirement

1.1. Personas

This document is primarily aimed at customers' Data scientists and Operations personnel responsible for the integration of ThetaRay Analytics Platform into their environment(s).

2. System Overview

The ThetaRay system ingests data consisting of financial transactions, KYC information and additional reference data.

Multiple protocols / formats, process the data and deliver alerts for investigation to either ThetaRay's own Investigation Center application or to external use case management applications. The diagram shown in Figure 1: below and following detailed list, illustrate (high level) the main interaction points of the ThetaRay system and broader ecosystem.

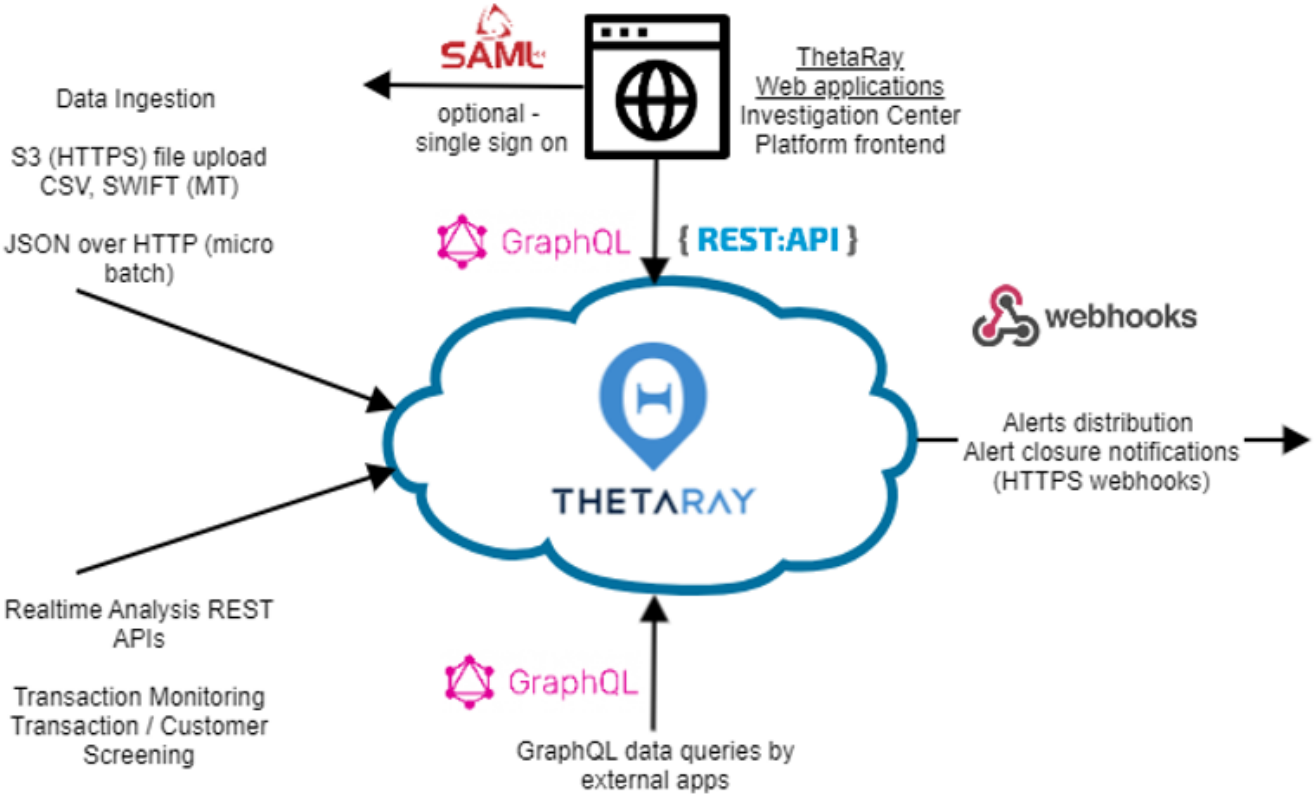

Figure 1: *The Main Interaction Points of the ThetaRay System and the Broader Ecosystem*

a. Data is ingested into the system either through file upload using an AWS S3 compatible protocol or RESTful API calls directed to an HTTPS endpoint exposed by ThetaRay. The system supports multiple data formats including delimited text files (CSV), and various flavors of SWIFT message formats when uploading data as files.

b. Realtime analysis of transactions / customers (transaction monitoring / sanction screening) is facilitated through REST APIs

c. The system analyses data and forwards any discovered alerts through HTTP web hook to external case management system.

d. Alerts can be investigated through ThetaRay's own Investigation Center.

e. Webhook based notifications can be set up to notify on alert creation / closure.

   i. Single sign on is supported for web based user interfaces through the use of SAML 2.0 / OpenID Connect protocols.

2.1. Main System Flows

The main flows of an operational ThetaRay environment include periodic model training activities, used to create an initial model as well as fine tuning the model over time as new data arrives and the detection flow, used to ingest new data, analyze it and generate alerts for identified risks.

The sections below (3.1.1) provide a high- level flow diagram of the activities running on each of these flows.

2.1.1. Detection Flow

The diagram below shown in Figure 2: illustrates the main flow used to generate alerts within the system.

- New data is ingested into the system in a designated area within a Minio Bucket. Data can be provided as delimited text files (e.g. CSV) or RJE files containing SWIFT MT and MX messages. Alternatively, micro batches of data can be fed into the system using an HTTPS endpoint exposing a RESTful API.
- Data acquisition tasks perform data parsing, validation & ingestion of data into datasets stored as Parquet files into Minio. These tasks run on ephemeral Pods on a Kubernetes environment. Datasets requiring interactive queries from the Investigation Center are either published to Postgres or stored in an alternative representation in Minio suitable for low latency, interactive queries.
- Once data is uploaded, transformed and launched as job, these jobs are run on ephemeral Kubernetes Pods hosting Spark Executors are read / write data of datasets in order to compute a set of computational features used for analysis.
- Prior to running detection a drift check is performed - trained model & reference dataset used for training are used for drift detection. In case drift is detected, an event is pushed towards the monitoring component.

- Detection runs as a Spark job of ephemeral Kubernetes Pods. The process loads a trained model from MLFlow's model repository, and invokes Algo prediction APIs using Spark UDFs in a scale out manner.
- Once detection is completed , Evaluated Activities are generated. These include data from the analysis Data Set enriched with the algorithmic scores and feature rating.
- The decision process applies a set of business rules to Evaluated Activities to identify whether a risk(s) are associated with individual activities. Identified risks are further transferred to ThetaRay's Investigation Center or to external case management systems.

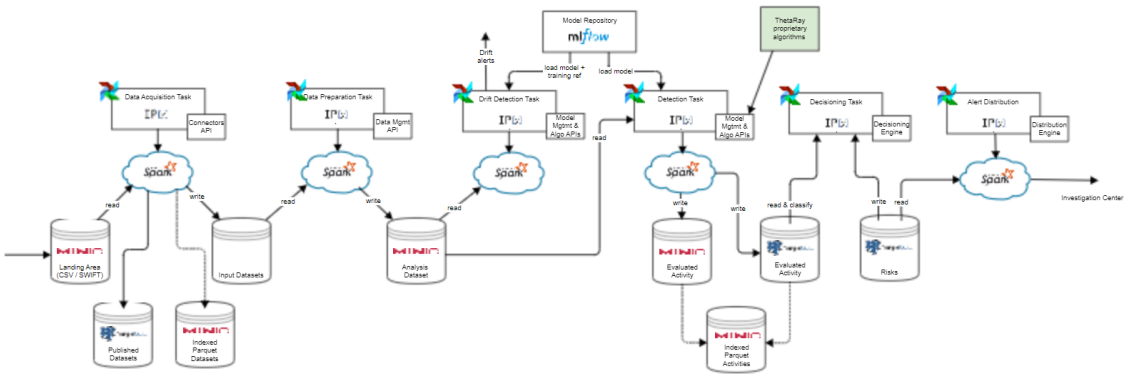

Figure 2: *The Main Flow (High Level) Used to Generate Alerts Within the System.*

2.1.2. Realtime Transaction Monitoring Flow

Real-time Transaction Monitoring flow is responsible for performing an evaluation of a single request (typically a transaction) delivered to the system through a REST API. The analysis of the request is performed synchronously and the response includes details of any alerts triggered on the given request.

To ensure low latency of request handling as well as enhancing the availability of the solution in presence of potential failures the flow is split into two parts:

1. Synchornous request handling – performs analysis of the incoming request by running a configurable set of rules and potentially perform read only access to data stores for fetching contextual data required by the rules. This allows leveraging query pre-planning as well as multiple read replicas.
2. Asynchronous processing – results of the synchronous part are queued and are asynchronously processed as micro batches (typically within a second) by consumer processes for long term persistent and alert creation.

The synchronous / asynchronous / database can be scaled independently based on expected the customer request volumes.

Additionally, the realtime flows are integrated with the system batch processing capabilities, allowing data computed in batch mode (such as monthly aggregates) to be exposed to the real-time components. This provides the ability to make decisions based on information that is computed by a combination of realtime and batch computed data reducing the need to perform all calculations in realtime.

The diagram below outlines the realtime flow:

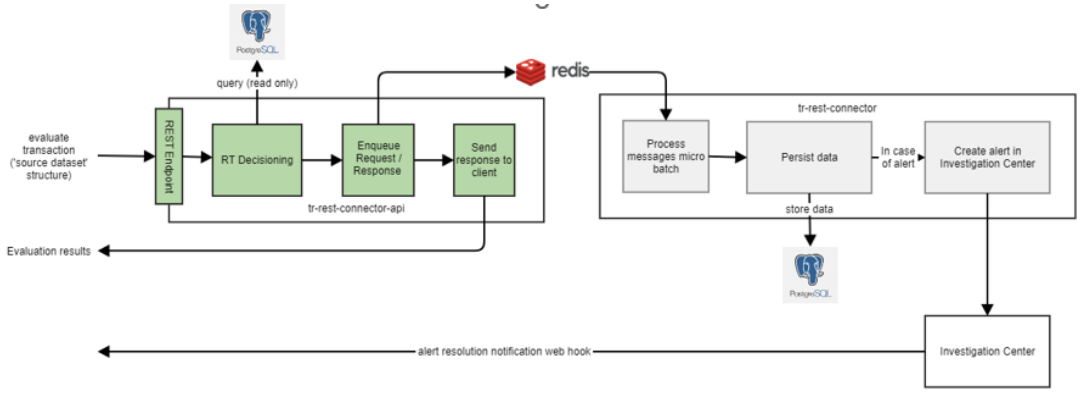

Figure 3: *Real Time Flow Image Diagram*

2.2. Model Training Flow

Model training flow results in a new / refined model for a given scenario as follows:

- The flow starts with computing a training data set for relevant historical data subset - this can be either using existing feature engineering strategy or an updated one.
- Once a data set consisting of engineered features is available the model training task is initiated. The training process runs on ephemeral Pods on the Kubernetes environment that includes the following main steps:
  - Training data set extraction through sampling
  - Automated hyper parameters optimization
  - Model training for selected features
- As a final step, a new model version is saved into the MLFlow model repository.

Model training (high level) is detailed as shown in Figure 4: below.

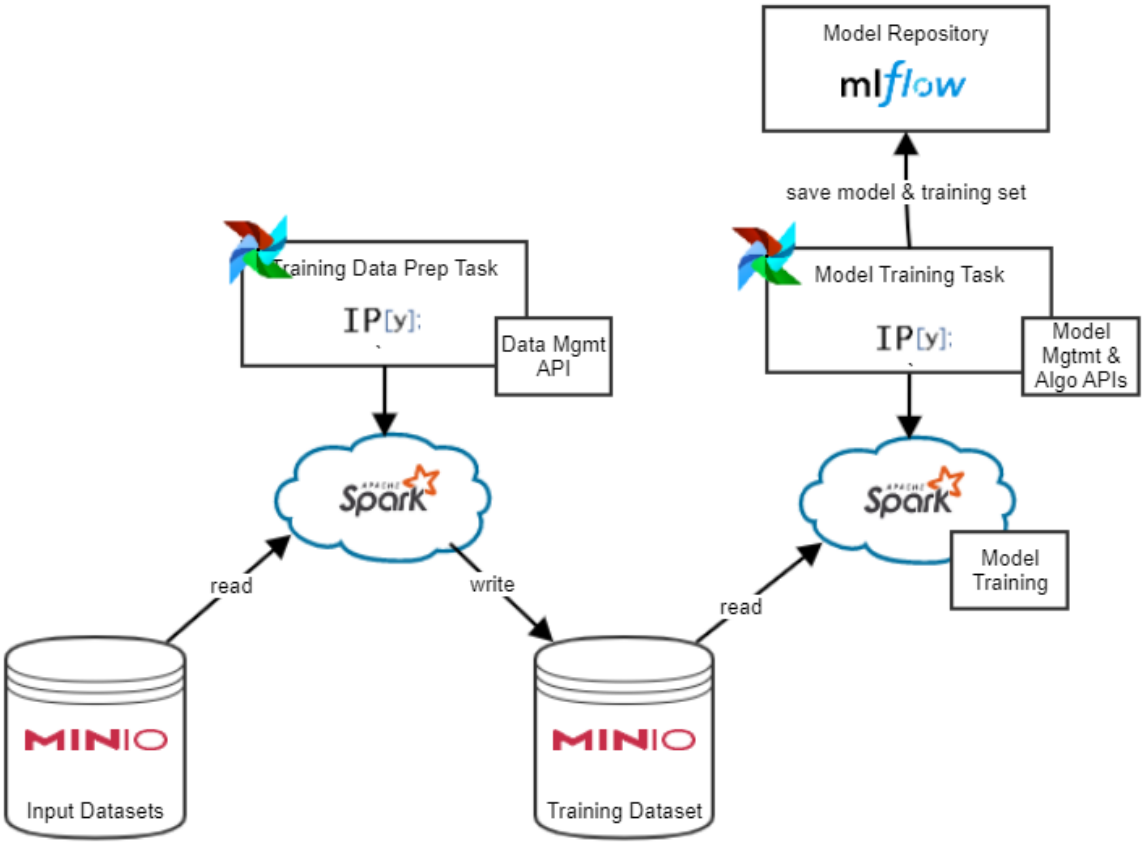

Figure 4: *Model Training System - High level*

2.3. Architecture Overview

ThetaRay system is composed of multiple services covering the functionality ranging from data stores up to web application that supports interactive investigation by analysts. The diagram shown in 2. below illustrates the overall system architecture. The following sections provide a high level description of the various sub-systems / components of the system.

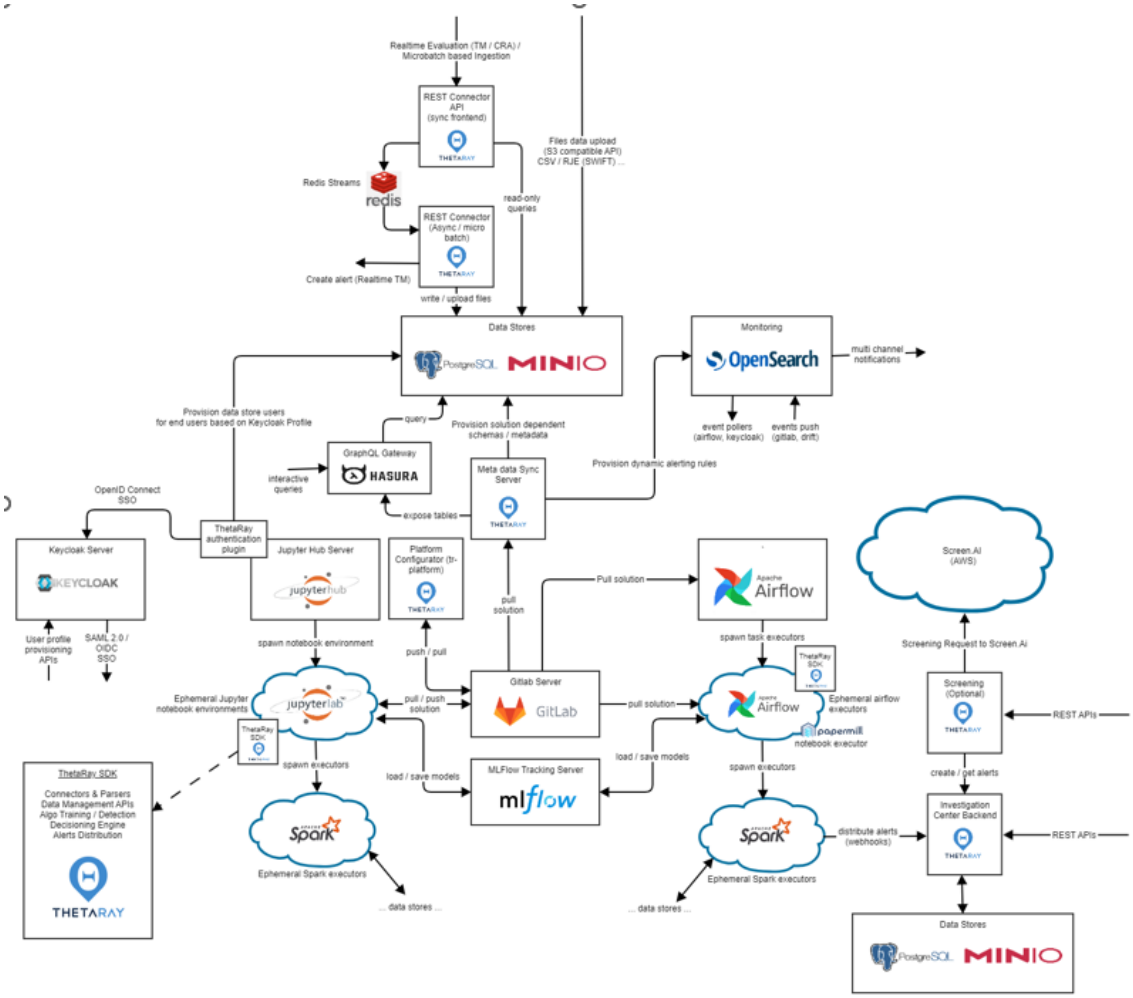

Figure 5: *Architecture Overview*

2.4. Inbound Data Feeds

Files consisting of data to be processed by the system can be uploaded through an AWS S3 compatible API, HTTPS endpoint. Any S3 compatible client (AWS command line tool, Minio MC or various S3 client SDKs such as Python boto3) can be used to upload data into the system. The system supports multiple file formats, configured through Connectors, which include:

- Delimited Text Files - CSV or other variants
- SWIFT MT with configurable translation rules

Alternatively, micro batches of data can be fed into the system using an HTTPS endpoint exposing a RESTful API.

2.5. Results Exposure

Analysis results are exposed through push / pull APIs as follows:

- Alerts can be distributed to user configurable endpoints through HTTPS based webhooks. The payload of the request consists of a JSON message of one or more alerts that includes activity data, associated risk and machine learning based scores and feature rating.
- A dynamic, GraphQL based interface can be used to access published data including:
  - Published Datasets
  - Analysis results (Evaluated Activities) and associated Risks

2.6. Solution Lifecycle Management

A Solution within the system includes multiple artifacts ranging from metadata definitions describing the various elements, notebooks / Python code consisting custom logic and risk definitions. All these artifacts are kept in a per tenant Git repository managed within an embedded Gitlab server, allowing versioning and Git driven solution deployment.

The Jupyter based interactive notebooks environment is pre-integrated with Git, allowing users to directly perform Git operations through an embedded user interface.

Moreover, solution deployment is fully driven by Git and is processed continuously, monitoring a dedicated Git branch holding artifacts staged for 'operationalization'. 'Metadata Sync' is used to control this functionality, and is used to update schemas, publish metadata into data stores, dynamically expose GraphQL APIs and finally, update automated processes with the updated solution.

2.7. Machine Learning Model Management

Machine learning models are managed within an MLFlow based Model Repository. The repository uses Postgres for metadata management and Minio for storing actual binary artifacts (e.g.. models and reference datasets that are used for drift detection).

Models are versioned within the repository and each model training process results in a new model version. Specific versions can be referenced by tags to allow full control of the version used by operational detection processes.

2.8. Batch Data Processing

Batch data processing activities in the platform include -

- Data upload into the platform through Connectors → fetching data from an external source, parsing it and ingesting the data into one or more datasets
- Data pre-processing and feature engineering jobs
- Model training and detection jobs

The amount of compute resources required for handling the above activities depends on volumes of data to be processed, job complexity and job completion deadlines. The resources required may exceed the capacity of a single server, requiring the platform to support distributed data processing both for enabling horizontal scaling of a single job, as well as support for executing multiple jobs in parallel.

The ThetaRay platform uses Apache Spark with a tight integration into the Kuberentes environment to enable distributed batch processing. Each job is executed over a configurable set of Spark Executors, each running within a dedicated Kubernetes Pod. The resource requirements of these Pods is configurable by a data engineer as part of the solution's implementation on a per activity basis. At the end of the activity, executors launched as part of it are automatically torn down allowing cluster / cloud resources to be reclaimed.

2.9. Workflow Automation

To automate the execution of one or more activities either triggered on demand or in a scheduled manner, Apache Airflow, a workflow automation tool focused on data processing is embedded into the system.

Airflow enables automation of Python code bundled within the Jupyter notebooks or arbitrary modules, each running within an Airflow Task, which is part of a broader DAG (Directed Acyclic Graph - Workflow). Each of these tasks runs within a dedicated Kubernetes Pod with a configurable specification, that is initiated upon task startup and torn down when completed.

2.10. Data Management

ThetaRay platform embeds 3 data stores for managing data handled by the system:

- Minio, An S3 compatible object storage deployment used to store raw data files uploaded to the system, as well as structured data used as part of the platform's batch data processing activities that also include data processing, feature engineering and algorithmic analysis. Data is stored in a columnar format, optimized for large scale batch processing

and is partitioned by time to allow for efficient data filtering and purge / archive operations.

- The Minio deployment spans multiple replicas, enabling horizontal scaling as well as high availability and resiliency.
- Postgres - Analysis results, Investigation Center alert data as well as any other data set that is available for interactive querying during investigation (transactions, customer information, etc.,) is persisted into Postgres. A configurable indexing scheme is used for each stored data element, and relevant tables are partitioned by time (at a monthly granularity) allowing the database to effectively store large amount of data and enabling data cleanup operations to be performed by deleting partitions. The database is deployed in a master / slave topology with an automated fail over process between master and slave(s), as well as the ability to scale out read operations across replicas (longer term road map)
- Elasticsearch - events storage for system events including audit of system operations and technical events such as task execution failures.

Data stores are deployed into a namespace that can be used across multiple tenants. Data segregation is supported for Minio through a Bucket per tenant,

2.11. Event Collection & Alerting

The system continuously collects events from the various components that make up the environment for auditing and application / data level health monitoring purposes. Depending on the component, events are collected either through polling relevant components (using Logstash) or by HTTP post requests. Events are delivered from:

- **Gitlab:** Tracking push operations into specific branches
- **Airflow:** Tracking changes to DAGS and execution status
- **Keycloak**: A user management application used to login events and configuration changes
- **Data / concept** drift events triggered by drift detection processes
- **Metadata sync** - changes to solution metadata and metadata distribution failures

These events are persisted into indexes within Elasticsearch allowing an administrator to explore the data through the Kibana user interface which is bundled with the system.

In addition, the system is setup to trigger alerts when detecting events indicating erroneous conditions. Alerting status can be viewed through Kibana,

and the system supports the configuration of push-based notifications through multiple channels including: Email, Slack and custom web hooks.

3. Security

3.1. Single Sign On

User facing applications utilize Keycloak as identity management and access control. OpenID Connect is used internally by all relevant components to implement SSO flows against a central Keycloak instance deployed as part of the system.

When running in standalone mode, users (including their credentials) and their profiles (roles / groups) are registered within Keycloak, Moreover authentication and password policy management is fully managed through the Keycloak administration console.

Typically, an integration with an existing identity provider is implemented. Integration is performed by configuring Keycloak, with the following options being supported -

- SAML 2.0 based SSO with optional roles / groups synchronization through SAML Attributes. No user passwords are exposed to the ThetaRay system and a web based login flow is used to redirect the user to the identity provider login page.
- LDAP based user federation - authentication requests are delegated to an LDAP / Active Directory server on the backend. Periodic / on demand job, synchronizes user groups / roles from LDAP into the ThetaRay system

3.2. Kubernetes / Openshift RBAC

The system uses multiple service accounts / roles on relevant Kubernetes / Openshift namespaces to enable the system to elastically spawn Pods and ensure system security in the presence of direct access to interactive notebook environments.

- Notebook environments and Airflow executors are spawned under a service account granted permission to spawn new Pods in a dedicated 'Tenant Executors' namespace
- Airflow Scheduler launching interactive Pods, is granted permission to spawn Airflow Executor Pods in the main tenant namespace.

The above approach ensures that the only Secrets accessible to end users working in interactive notebook environments are the ones injected to the Jupyter Pod and include auto generated credentials assigned to the specific end users with the correct permissions.

3.3. Data at Rest Encryption

Environment Type	Kubernetes Flavor	Encryption Method
SaaS – Azure	AKS	Managed disks encryption, with encryption keys either managed by Microsoft or ThetaRay / customer (BYOK)
OpenShift - Azure	OpenShift	Managed disks encryption, with encryption keys either managed by Microsoft or customer
OpenShift – On-prem	OpenShift	Deployment specific – dependent on selected storage method
On prem - Rancher	Rancher RKE	Block device encryption using LUKS / dm-crypt. Setup is the customer's responsibility

In addition to encrypting data at storage level, ThetaRay provides applicative data at rest encryption of sensitive information for Azure based on deployments. This functionality relies on an integration with the Azure Key Vault to manage a master Key Encryption Key (KEK), which can either be managed by the customer or by ThetaRay.

Applicative data at rest encryption provides end to end data encryption, which is tightly integrated with system components to allow role-based access control of who can be granted access to data. Moreover, when encryption is enabled, administrators having direct access to databases or object storage, are no longer able to view sensitive PII information. This type of information will appear in encrypted form when queried.

Encryption at rest covers the various phases at which data is managed by ThetaRay –

1. Data uploaded into the landing bucket in object storage can be fully pre-encrypted by the customer using a per file data encryption key.
2. Managed datasets within the system located either in object storage (Minio) or relational database (Postgres), store sensitive personally identifiable information (PII) in encrypted form. The set of columns to be encrypted is configurable and should be agreed upon during project implementation.
3. Audit events that hold customer details and are persisted into OpenSearch, store this information in encrypted form.
4. If audit events are exported to the customer through a bucket in ThetaRay's object storage, then individual audit messages are fully encrypted.

To allow encryption key rotation, without requiring full data rewrites, the system relies on a Fix-Data En-cryption Key. The latter is established during system setup and a Key Encryption Key that is managed in Azure Key Vault and can be freely rotated. No copies of the Data Encryption Key are stored in plain form. The Data Encryption Key is stored in encrypted form. It is encrypted through the Key

Encryption Key man-aged by the Azure Key Vault. The Key Encryption Key never leaves Azure Key Vault, and these actions take place through facilities provided by Azure Key Vault (wrap key / unwrap key).

3.4. Data in Transit Encryption

All ingress / egress traffic into / out of the ThetaRay Kubernetes environment is HTTPS based and is encrypted using TLS.

HTTPS / TLS termination is handled by a Kubernetes level ingress controller and is encrypted / authentication using certificates associated with the domain name under which the ThetaRay system is installed.

Egress traffic from the cluster, used to push notification to remote HTTPS endpoints, is authenticated through configurable credentials and is encrypted using TLS.

Starting from release 6.1, all intra cluster traffic is encrypted using TLS 1.3. Certificates are dynamically issued at deployment time (initial installation / env update) for all relevant components and are signed using a 'self signed' Certificate Authority, with a certificate / private key registered as a deployment level parameter. The certificates validity time is two years and certificates are automatically regenerated on every deployment level activity.

4. Highly Available Deployment on Openshift/ Azure

For ensuring high availability in presence of either component or data center failures the ThetaRay system supports deployment of the system into an Openshift cluster running on Microsoft Azure which spans 3 availability zones. Deploying across 3 availability zones enables the system to continue to operate in presence of a whole available zone failure.

When deployed across multiple Availability Zones, the system is set up to replicate data across different data zones to ensure availability, should a zone failure occur. The replication mechanism used by each stateful component is dependent on the specific component and may utilize applicative mechanisms or Azure's ZRS (Zone Redundant Storage) to replicate the data. Stateless components are deployed in multiple replicas which spread across multiple availability zones.

Following is a list of stateful components and their approach for dealing with high availability and data replication:

- Postgres - 2 replicas with asynchronous streaming replication setup between them
- Minio - 8 (or more) replicas spread across 3 availability zones (3-3-2 topology) with built in data redundancy
- Gitlab - 3 instances of Gitaly with built in data replication of Git repositories
- Opensearch - 3 replicas spread across 3 Availability Zones with built in replication
- Keycloak - 2 instances deployed across 2 availability zones with built in clustered cache
- Logstash - minimal state stored on volume in ZRS mode
- Redis - minimal state stored on volume in ZRS mode

5. Deployment

5.1. Overview

The ThetaRay system is packaged and is deployed into a Kubernetes / Openshift environment through a set of Helm charts invoked through the Helmfile tool. The following Kubernetes environments are supported -

- RedHat Openshift 4.12 / 4.14 (on-prem / Azure)
- Azure Kubernetes Service (AKS)
- Bundled Rancher RKE - supported for existing deployments only. Not available for new deployments

The system uses a mixture of stateless / stateful containers and such requires the Kubernetes environment to support dynamically provisioned Persistent Volumes allocated within centralized block storage. In case of a running on a Rancher RKE based Kubernetes environment, ThetaRay deploys Longhorn - a storage platform which uses local disks to provide resilient Kubernetes Persistent Volumes.

The following sections detail the structure of the deployment across Kubernetes namespaces and the sizing requirements for the various components.

5.2. Deployment Structure

ThetaRay 6.0 is deployed across multiple Kubernetes namespaces, allowing multiple 'tenants' to share a common infrastructure. Four main namespace 'types' are used by the system:

- Shared infrastructure namespace - hosts infrastructure components that serve multiple tenant environments and includes components such as Minio, Postgres, Gitlab, Keycloak, Opensearch and Hasura GraphQL Gateway
- Per tenant's solution, two namespaces (the separation into two namespaces is used to ensure that users that have access to notebook environments that dynamically launch executor Pods, do not have any to secrets managed as part of the tenant's namespace)
- Main namespace used to host components that are continuously running, including Jupyterhub, MLFlow & Airflow as well as dynamic Jupyter notebook environments
- Executors namespace used to launch dynamic Airfow / Spark executors.

- Application namespace - hosts an Investigation Center instance and may access data from multiple platform tenants through a GraphQL Gateway (Hasura)

The diagram shown in Figure 6: below illustrates the deployment topology described above.

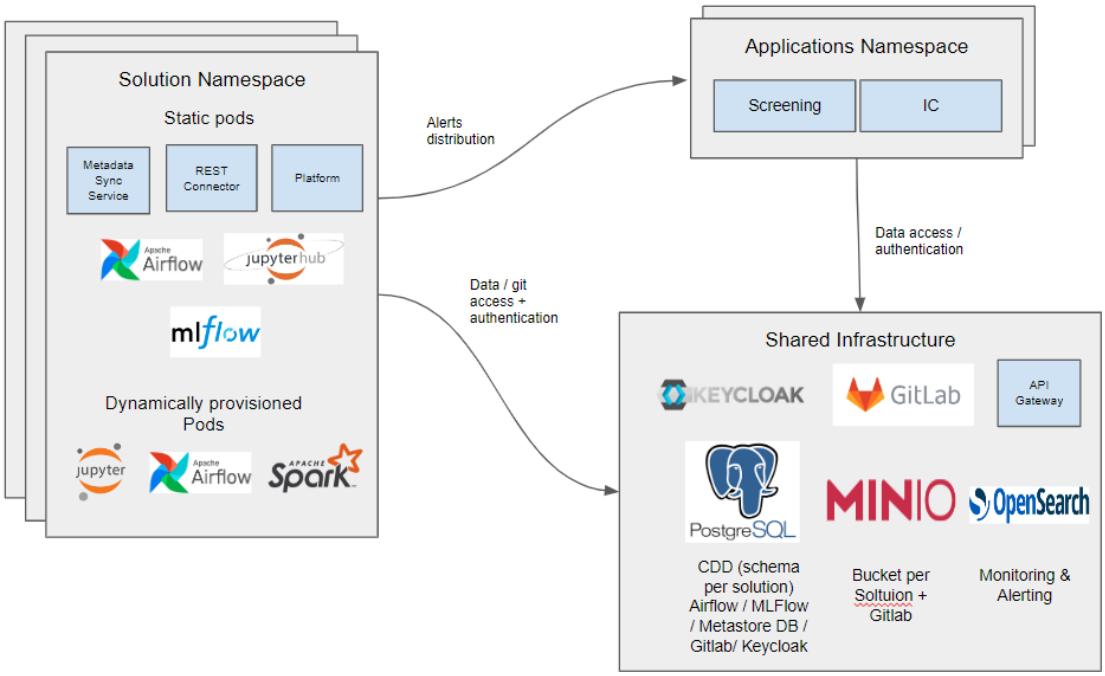

Figure 6: *Deployment Structure - High Level*

5.3. Deployment Appendix

The following sections detail the components that are part of each of the namespace and their resource requirements.

5.3.1. Shared Infrastructure Namespace

Component	Description	Pods
Data Access Service	Low latency queries of Parquet Files, data-access-service	data-access-service
PostgreSQL	Relational database used as ThetaRay's Central Data Deplot, as well as for MLFlow / Gitlab / Airflow / Hive Metastore database storage.	postgres-*
Minio	Distributed object storage used for raw data uploaded to the system, and structured data persisted in the system	minio-*

Component	Description	Pods
	and used within batch processing activities. Additionally → u	
Opensearch	Central application event collection and monitoring stack. Includes Elasticsearch for event collection and alerting + Kibana for visualization / exploration capabilities	opensearch-cluster-master-*, opensearch-dashboards-*
Hasura	Stateless GraphQL gateway, enabling interactive data access to ThetaRay managed data in the Central Data Depot (Postgres)	
Logstash	Event collection middleware (by defa	logstash-logstash-0
Redis	Caching service (Gitlab / session management)	redis-master-0
Gitlab	Central version control system for solution artifacts	gitlab-gitay-* gitlab-registry-* gitlab-sidekiq-all-in-1-* gitlab-task-runner-* gitlab-webservice-default-*
Keycloak	Central security & identity management	keycloak-*
PGAdmin4	Postgres administration web application	pgadmin4-*
API Gateway	central API gateway directing REST API calls to underlying components	tr-api-gateway

5.4. Tenant Components

Static Deployments / Statefulsets

Component	Description	Pods	Scalability & Availability
Airflow	Scheduling & orchestration of data processing / analysis jobs	airflow-scheduler-* airflow-webserver-*	One or more instances of components for high availability and scalability purposes Scheduler uses database level locks to ensure consistency in presence of multiple scheduler instances. Webserver is a stateless web application
Jupyterhub	Multi user interactive Jupyter notebook environment	proxy-* hub-*	Single instance, with automated fail over at Kubernetes level. During fail over, notebook environments cannot be accessed.
MLFlow	Experiments tracking and ML	mlflow-*	Stateless service with multiple instance (single instance by default). In case of

Component	Description	Pods	Scalability & Availability
	model repository		unavailability, models cannot be published and detection cannot be performed.
Logstash	Event collection middle ware from tenant's environment	platform-logstash-0	Single instance with automatic fail over at Kubernetes level
spark-history-server		spark-history-server-*	Single instance - fail over through Kubernetes
tr-metadata-sync			
REST Connector	Data ingestion REST API endpoints	tr-rest-connector	multiple replicas
Platform Services	platform management APIs such as rules parameters editing	tr-platform	Kubernetes based failover

5.4.1. Dynamically Provisioned Pods

Component	Description	Pods
Jupyter Notebook Environment	Single user interactive notebook environment	jupyter-<username>
Airflow Executors	Per task Pods, executing Jupyter notebooks / python code defined by the user (including launching Spark drivers)	custom
Spark Executors	Per Spark executor Pod	custom

5.4.2. Applications - Screening / IC

Component	Description	Pods
Investigation Center Backend	Investigation Center backend APIs	tr-icbe
Investigation Center Frontend	Investigation Center static resources and request forwarding to backend	tr-icfe
Screening	Screening APIs exposure and orchestration	tr-screening
Central Data Depoy	Service supporting data synchronization of the CRA for the Investigation Center	tr-CRA

5.5. Additional Requirements - Hardware

Note: When deploying ThetaRay it is a hardware requirement to use only solid state devices (SSD).

6. Backup and Disaster Recovery

To be able to ensure system and data availability in scenarios such as whole data center (region) unavailability for a significant amount of time or data corruption due to technical or user errors the system provides the following capabilities:

- Periodic, scheduled data backup to external storage - a process that backups up the data managed by the system to a storage device running outside of the Kubernetes / Openshift environment. Backups are used to protected against data corruption due to human error or technical failures.

  Currently the following backup targets are supported:

  - Azure BLOB Storage - for environments running on Openshift / Azure
  - Google Cloud Storage - for environments running on Google Kubernetes Engine
  - S3 Compatible Storage

- Continuous data replication from an active environment on a primary data center / region to a passive, fail over site. The mechanism enables starting an environment on a fail-over site using replicated data, and requires a running Minio cluster on the remote site to capture and persist replicated data (Minio running over Kubernetes / Openshift).

  The mechanism is agnostic to the underlying infrastructure and works on both on-prem / cloud environments with the only assumption being is the availability of network connectivity between the sites.

  The replication facility performs continuous replication of both Postgres and Minio data across data centers. Replication of monitoring information in Opensearch and data within the Gitlab repository are replicated using daily snapshots.

  The following diagram shows the Backup and Disaster Recovery in a high level.

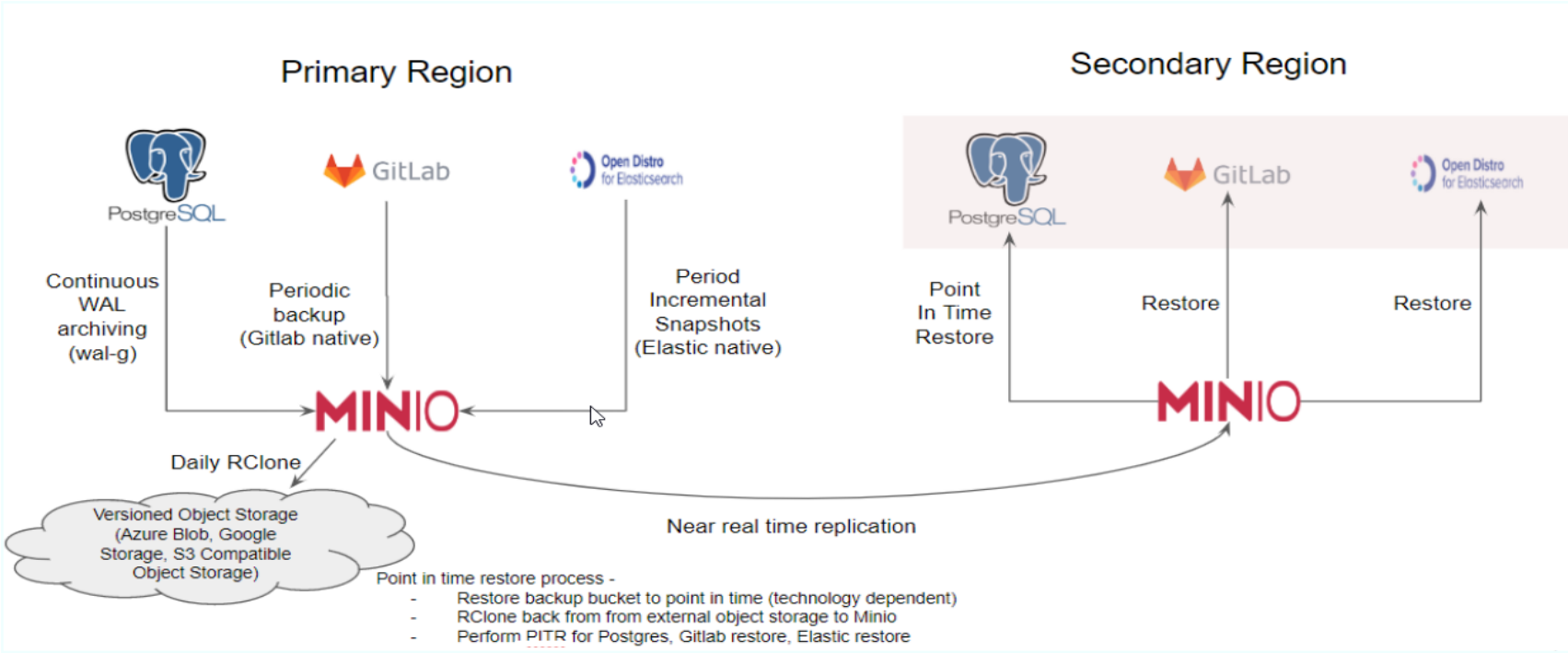

Figure 7: *Overview of the Backup and Disaster Recovery Architecture*

7. SaaS Deployment

7.1. SaaS Deployments on Azure

Overview

The ThetaRay system can be provided as a fully managed SaaS offering running on Microsoft Azure. A dedicated environment is deployed for each customer (tenant) – this includes the compute and storage resources (managed disks / storage account).

The Thetaray environment is deployed within a primary Azure region which is selected based on customer's security and regulatory requirements. Backups, and an optional disaster recovery environment are deployed into a secondary site which, as with the primary site, is selected based on customer requirements.

By default the following regions are used:

- For Americas (NA/LATAM) the primary site is US East (Virginia US) and the backup site is West US (Washington State US)
- For Europe and MEA - the primary site is West EU (Netherlands) and backup site is North EU (Ireland)

Deployment into other regions is possible, based on availability of Azure datacenters in the relevant location(s)

The following diagram illustrates a single tenant deployment structure

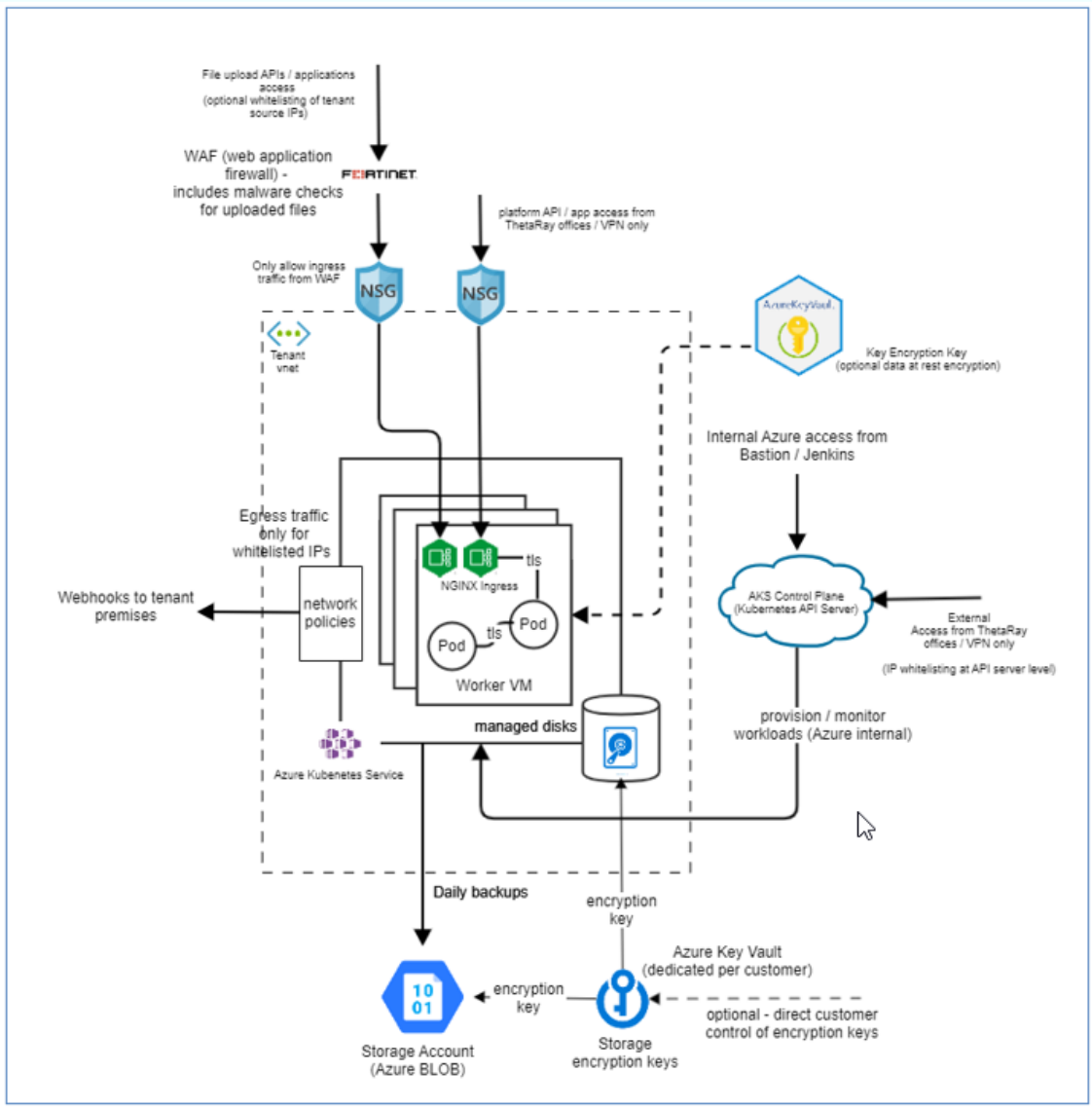

Figure 8: *SaaS Single Tenant Deployment Architecture*

Networking

- Only HTTPS based traffic is allowed to arrive from the public internet / customer premises and the traffic must go through a web application firewall (currently Azure Application Gateway)
- Internal traffic inside the cluster is TLS based and encrypted through a self signed CA
- All ingress / egress traffic from the environment is HTTPS / TLS based
- SSH access to a bastion server is only granted to whitelisted IPs (ThetaRay office / VPN)
- Direct access to the Kubernetes API is granted only from well known IP ranges (ThetaRay offices / VPN / bastion / CI-CD infrastructure)

- External network access from the environment is only granted to whitelisted IPs

Data at Rest

- Data at rest is encrypted through disk level encryption, provided by Azure with encryption keys managed by Azure
- Optionally, applicative data encryption can be implemented. The Azure Key Vault is used to hold a master Key Encryption Key which is used to protect data encryption keys used for data encryption / decryption.

Vulnerabilities Management

- All software packages and configuration go through continuous scans for vulnerabilities (CVEs / miss-configuration) with a CSPM tool (Wiz)
- Issues should be resolved according to ThetaRay's Vulnerability Management Procedure