elastic

Search. Observe. Protect.

# Engagement Report

## Directorate General of Analytics and Risk Management (DGARM)

## Design Document

2025 August 29th

Salim Alaeddine, Mattias Brunnert

elastic.co

# Table of Contents



elastic

# Document Properties

## Version History

| Version | Date | Author | Description |
|---|---|---|---|
| 1.0 | 2025 August 18th | Salim Alaeddine | Initial draft |
| 1.1 | 2025 August 26th | Mattias Brunnert | Search Design |
| 1.2 | 2025 August 29th | Mattias Brunnert, Salim Alaeddine | Review |

## References

| Ref. No | Document Name | Location | Description |
|---|---|---|---|
| 1 | | | |

# Executive Summary

## Overview

The Directorate General of Analytics and Risk Management (DGARM), functioning under the Central Board of Indirect Tax and Customs (CBIC), provides intelligence inputs and leverages big data analytics to assist tax officers in policy formulation, compliance monitoring, and detection of tax evasion. In alignment with these objectives, Infosys and the TeTrain team have engaged Elastic Professional Services to design, validate, a Search and Observability solution. This engagement represents the first-time deployment of an Elastic architecture within DGARM, with the goal of ensuring best practices are embedded into the platform from inception.

The scope of work covers:

- Elastic Platform Design & Best Practices
- Architecture design and design review
- Pipeline and ingestion recommendations and patterns
- Elastic cluster health check and validation against the agreed Design Document
- Review of cluster configuration, sharding strategy, lifecycle management policies, and ingestion strategy

## Participants

| Org | Name | Role/Position | mail |
|---|---|---|---|
| Tetra | Biswajit Banerjee | Project Manager | biswajit@tetrain.com |
| | Deepa | Leads the software team | deepa@tetrain.com |
| | Ravinder Kuma | | ravinder.kumar@tetrain.com |
| | Sweta | | sweta@tetrain.com |
| | Takshay | | takshay@tetrain.com |
| | Tushar | | tushar@tetrain.com |
| | Rakeshdubey | | rakeshdubey@tetrain.com |

| | | | |
|---|---|---|---|
| | Mukul | | mukul@tetrain.com |
| Infosys | Muralidharan | Lead Architect | muralidharan_S12@infosys.com |
| | Bhasha Sisodia | Operations Manager | bhasha.sisodia@infosys.com |
| | Karan Neelkanth | Platform Lead | karan.neelkanth@infosys.com |
| | Divya Dhyani | | divya_dhyani@infosys.com |
| | Bimlesh Kumar | | bimlesh.kumar@infosys.com |
| | Ashutosh Tiwari | | ashutosh.tiwari13@infosys.com |
| | Krithika Chakrapani | | krithika_chakrapani@infosys.com |
| | Vikram K | | vikram_k@infosys.com |
| | Anant Pendse | | anant_pendse@infosys.com |
| Elastic | Salim Alaeddeine | Consulting Architect | salim.alaeddine@elastic.co |
| | Mattias Brunnert | Principle Consulting Architect | mattias.brunnert@elastic.co |
| | Sri Suba Selvachamy | Delivery Manager | sri.selvachamy@elastic.co |

## Business Value

By adopting Elastic as the foundation for its analytics and observability strategy, DGARM can realize significant value:

- **Enhanced Data-Driven Decision Making**: Empower timely intelligence and actionable insights.
- 
- **Operational Efficiency**: Centralized log, metric, and trace visibility improves system monitoring, enabling faster root-cause analysis and reduced downtime.
- **Improved Compliance and Enforcement**: Strengthened ability to detect anomalies, identify patterns of evasion, and support investigations.

- **Future-Ready Architecture**: A design aligned with Elastic best practices ensures scalability, resilience, and adaptability to future data growth and regulatory needs.

## Outcomes

The engagement with Elastic Professional Services is expected to deliver:

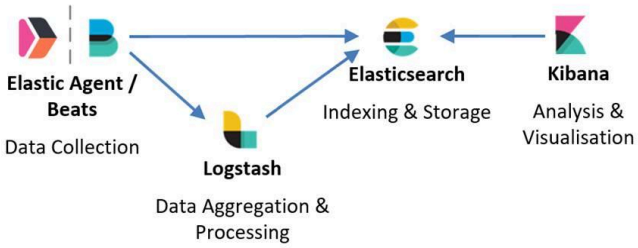
- A validated Elastic architecture design, aligned with business requirements and technical best practices.
- Lifecycle and sharding strategies optimized for performance, cost efficiency, and data retention requirements.
- A health check and validation, identifying risks, gaps, and remediations for the current deployment.
- Increased stakeholder confidence in the Elastic platform as a secure and reliable foundation for DGARM's big data analytics initiatives.

# Solution Overview

## Elastic Stack Overview

An Elastic Stack deployment consists of several core components, each designed to support specific roles in the collection, processing, storage, analysis, and visualization of observability data. These components work together to deliver a scalable, flexible, and feature-rich monitoring and analytics platform.




Elastic Stack can be deployed in various forms, including fully managed Elastic Cloud Service, self-managed orchestrated deployments, or standalone on-premise clusters. In this engagement, DGARM will deploy a standalone on-poremise cluster in their local data center. The following provides a brief description of each component and its role in an Elastic Stack deployment.

### Beats

Beats are lightweight data shippers installed on edge devices or servers to collect and forward data to Elasticsearch or Logstash. There are different types of Beats, each tailored for specific data sources:

- **Filebeat**: Monitors and ships log files.
- **Metricbeat**: Collects system and service-level metrics.
- **Auditbeat**: Collects audit data from Unix-based systems.
- **Winlogbeat**: Captures Windows event logs.
- **Packetbeat**: Monitors network traffic and packets.

Each Beat offers a set of predefined modules for common data sources and supports custom configurations for advanced use cases.

## Elastic Agent

Elastic Agent consolidates the functionality of multiple Beats into a single, unified agent. Managed centrally through Fleet Server, it simplifies deployment, scaling, and lifecycle management across diverse environments. Elastic Agent supports a wide array of out-of-the-box integrations for logs, metrics, and security telemetry. For a full list of integrations available see Elastic Agent Integrations.

## Logstash

Logstash is a powerful data processing pipeline tool that supports the ingestion, parsing, enrichment, transformation, and routing of data. It uses pipelines composed of input plugins, filter plugins, and output plugins to handle a wide variety of data sources and formats. This makes Logstash ideal for complex or custom data processing workflows before the data reaches Elasticsearch.

## Elasticsearch

Elasticsearch is the heart of the Elastic Stack. It serves as a distributed, RESTful search and analytics engine designed for scalability, speed, and relevance. It provides the storage, indexing, and powerful search capabilities that drive real-time visibility and insights from ingested data.

## Kibana

Kibana acts as the frontend interface for the Elastic Stack. It allows users to visualize, explore, and interact with data stored in Elasticsearch. Kibana supports dashboards, alerting, lens visualizations, drilldowns, and many plugins for advanced use cases such as SIEM, APM, and observability.

## APM / Fleet Server

The APM (Application Performance Monitoring) component provides performance insights and distributed tracing for monitored applications. Fleet Server is the centralized orchestration layer that manages Elastic Agents, ensuring policy-driven configuration, agent status tracking, and seamless integration management across the stack.

# Cluster Design

## Operating System

The Elastic Stack can be installed on Windows or Linux operating systems. Linux is the most commonly used operating system and therefore has a more comprehensive knowledge base for configuration and troubleshooting. DGARM aims to deploy Elastic stack on Openshift VMs.

The complete Elastic Stack support matrix can be found here.

## Hardware Requirements

This section details the recommended hardware requirements for Elastic Stack components. The sizing and recommended topology is provided in the Elastic Deployments section. Ingest, Transform and Remote Cluster Client nodes have not been included as these are typically not dedicated nodes.

| Component | RAM | CPU | Storage (SSD) | Disk Performance |
|---|---|---|---|---|
| Elasticsearch - Master | 64GB max | 1 CPU per 2GB RAM | 32GB storage total | Highest IOPS (10K+) |
| Elasticsearch - Data Hot | 64GB max | 1 CPU per 2-4GB RAM | 30GB storage per 1GB RAM | Highest IOPS (10K+) |
| Elasticsearch - Data Warm | 64GB max | 1 CPU per 4GB RAM | 100 GB storage per 1GB RAM + Snapshot repository | Highest IOPS (10K+) |
| Elasticsearch - Machine Learning | 64GB max | 1 CPU per 2GB RAM | 32GB storage total | Any |
| Kibana | 16GB max | 1 CPU per 2GB RAM | 32GB storage total | Any |
| Logstash | 32GB max | 1 CPU per 2GB RAM | 32GB storage unless Persistent Queue (PQ) is required | Medium IOPS (7K+) if PQ is used, otherwise any. |
| Elastic Agent | 16GB max | 1 CPU per 2GB RAM | 32GB storage | Any |

## Network Connectivity

As DGARM is deploying a new cluster, the following connections may be required for the Elastic Stack components.

| Flow | Source | Destination | Default Port |
|---|---|---|---|
| Agent Management | Elastic Agent | Fleet Server | TCP/8220 |
| Elastic Agent Ingestion via Logstash | Elastic Agent | Logstash | TCP/5044 |
| Elastic Agent Ingestion to Elasticsearch | Elastic Agent | Elasticsearch | TCP/9200 |
| Agent Policy Retrieval | Fleet Server | Elasticsearch | TCP/9200 |
| APM Data Ingestion | Elastic APM Agent | APM Server | TCP/8200 |
| Kibana User access | User | Kibana | TCP/5601 |
| Elastic agents Artifacts | Elastic Artifact registry | Elastic Artifact registry | TCP/443 |

## Security - SSL/TLS Certificates

Elastic self-managed deployments require an initial security setup to enable features such as encrypted communications, user authentication, and secure node enrollment. This includes configuring Transport Layer Security (TLS) for both the HTTP and transport layers, setting passwords for built-in users, and generating enrollment tokens to securely connect Kibana and additional Elasticsearch nodes.

Elastic supports two approaches for performing this initial security bootstrap: automatic and manual setup. Securing Kibana always requires manual configuration to complete the integration.

Elastic uses TLS certificates to secure communications in two primary layers:

- HTTP Layer: Encrypts traffic between clients (e.g., users, Kibana) and Elasticsearch nodes, ensuring data in transit is protected.

- Transport Layer: Encrypts inter-node communications within the cluster and between clusters (if applicable), providing mutual authentication and preventing unauthorized nodes from joining.

Types of TLS certificates required and their usage

| Certificate Type | Service/Node | Usage |
|---|---|---|
| Certificate Authority (CA) | All nodes | Used to sign and trust all other TLS certificates across the deployment. |
| Elasticsearch Transport Certificate | All Elasticsearch nodes | Mutual TLS for inter node communication |
| Elasticsearch HTTP certificates | All Elasticsearch nodes | TLS for any client making API/HTTPs connectivity |
| Kibana Certificate | Kibana nodes | TLS for any end user accessing Kibana using browser |
| Fleetserver Certificate | Fleetserver node | TLS for any agent accessing fleet server |

DGARM may choose to use their own TLS certificates by adopting the manual setup approach.. It involves enabling different layers of protection in sequence, depending on your security requirements.

1. Configure transport TLS: Required for multi-node clusters running in production mode. Secures communication between nodes and prevents unauthorized nodes from joining the cluster.
2. Configure HTTP TLS: Secures all client communications over HTTPS, including traffic between Kibana and Elasticsearch, and between browsers and Kibana. Recommended for all clusters, even single-node setups.

For additional TLS configuration options, refer to Manage TLS encryption in self-managed deployments.

## Cluster Resiliency

As a distributed system, Elasticsearch is capable of tolerating failure even if some of the components are not available. Elasticsearch accomplishes this through redundancy of every component, including the data stored. A resilient cluster must have the following characteristics:

- At least three master-eligible nodes
- At least two nodes of each used role
- At least one replica shard for every primary shard, unless the index is a searchable snapshot ( the data is backed up in the snapshot repository)

The above configuration is the minimum requirements for resiliency i.e. the cluster will stay operational if a single node fails. This concept can be expanded for further resiliency by dividing the cluster into availability zones. Each availability zone should have independent power supply and other supporting infrastructure. In practice, this may be a different fault domain within a data centre or even another data centre entirely.

By leveraging availability zones, the cluster remains functional even if an entire availability zone becomes unavailable. For a cluster to survive an availability zone failure, the following characteristics must be present:

- At least three master-eligible nodes, each situated in a different server racks
- A symmetrical deployment of each node role across at least two server racks
- At least one replica shard for every primary shard, unless the index is a searchable snapshot spread across at least two availability zones using shard allocation awareness.

## Elastic Deployments

DGARM is planning to deploy the Elastic clusters locally within their primary datacenter, maintaining a single physical site and single availability zone architecture.

This approach introduces a known risk, any failure impacting the primary datacenter, such as power outages, hardware failures, or connectivity loss, could result in a full cluster outage, affecting service

availability and access to data. DGARM acknowledged and accepted the inherent risk associated with a single availability zone deployment.

To mitigate the risk of data loss and to support business continuity objectives, DGARM has planned an active–passive disaster recovery (DR) strategy:

- A passive cluster will be deployed in the DR site.
- Snapshots from the primary site will be replicated to the DR site.
- In the event of a disaster, a manual failover process will be initiated, restoring data from the most recent snapshots.
- Disaster recovery strategies and procedures will be enforced and periodically reviewed to validate recovery time (RTO) and recovery point objectives (RPO).

While this approach does not provide instantaneous failover or cross-site replication (as in multi-AZ or cross-cluster replication designs), it aligns with DGARM's current infrastructure, budget, and licensing constraints, providing a balance between risk management and operational feasibility.

## Cluster topology

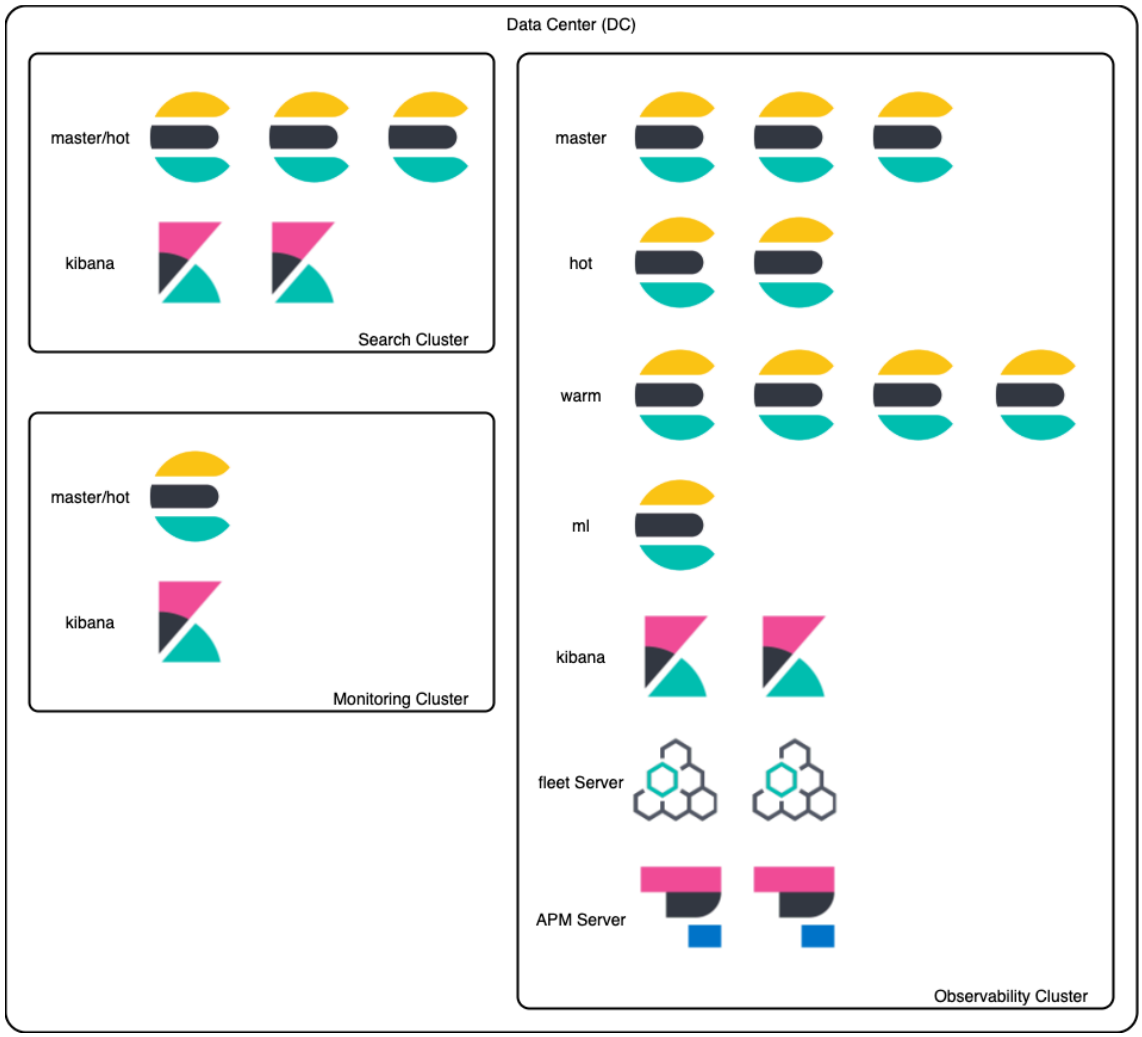
Initial sizing for the Elastic deployment was carried out during presales discussions between the Elastic Solution Architects and the client. Based on this exercise, a single unified production cluster was originally proposed to support both Search and Observability use cases.

Following best practices and further refinement during discovery workshops, the design was adjusted to better address the criticality of the workloads, the service-level requirements (SLAs), and the need for resilience and operational visibility. As a result, the topology was updated to include:

- Dedicated Search Cluster – to handle search-intensive workloads independently.
- Dedicated Observability Cluster – to ingest and analyze logs, metrics, and traces without impacting search performance.
- A minimal Monitoring Cluster – to provide end-to-end visibility of the production clusters' performance and health.

The overview topology for each cluster can be found in the diagram below




## Search Cluster

| Role | CPU | Memory (GB) | Disk (SSD) | Quantity |
|---|---|---|---|---|
| Hot ingest and Master Nodes | 16 | 64 | 2TB | 3 |
| Machine Learning | 8 | 64 | 200GB | 1 |
| Kibana | 8 | 16 | 200GB | 2 |

## Observability Cluster

| Role | CPU | Memory (GB) | Disk (SSD) | Quantity |
|---|---|---|---|---|
| Master Nodes | 8 | 16 | 200 GB | 3 |
| Hot ingest Nodes | 16 | 64 | 2TB | 2 |
| Warm nodes | 16 | 64 | 5 TB | 4 |
| Machine Learning | 16 | 32 | 200 GB | 2 |
| Kibana | 8 | 16 | 200 GB | 2 |
| Fleet Server | 8 | 16 | 200GB | 2 |
| APM Server | 8 | 16 | 200GB | 2 |
| Snapshot Repository | - | - | - | |

DGARM acknowledges the license limitations may cause performance impact to the cluster. DGARM will closely monitor cluster performance and scale up and out if required.

## Monitoring Cluster

A dedicated monitoring cluster will be deployed to collect the logs and metrics generated by the Elasticsearch clusters. The monitoring cluster is critical to all production grade Elasticsearch deployments. As part of best practice, a 3-node monitoring cluster is recommended to ensure high availability, resilience, and fault tolerance. However, to align with the current procured license entitlement, the monitoring cluster has been initially deployed as a single-node cluster to provide the minimum monitoring capabilities required. The client will revisit this approach in future licensing and scaling exercises to align the monitoring cluster with high-availability standards.

Below is the desired topology

| Role | CPU | Memory (GB) | Disk (SSD) | Quantity |
|---|---|---|---|---|
| Master/Hot Data Nodes | 4 | 8 | 1 TB | 3 |
| Kibana | 2 | 4 | 200 GB | 2 |

However to stay within the license, the initial cluster size could be as follow

| Role | CPU | Memory (GB) | Disk (SSD) | Quantity |
|---|---|---|---|---|
| Master/Hot Data Nodes | 4 | 8 | 1 TB | 1 |
| Kibana | 2 | 4 | 200 GB | 1 |

## Placement of Elasticsearch Nodes

To increase resiliency of the Elastic stack it is important to place virtual machines so that they don't share common hardware resources, including server rack, power supplies, etc. It is especially important that each stack component do not share hardware resources within the same cluster, including:

- Master nodes
- Hot data nodes
- Warm data nodes
- Machine learning nodes
- Kibana
- Fleet

### Shard Allocation Awareness

Once the cluster grows to a larger size and not sharing hardware between nodes becomes impractical, Elastic recommends DGARM implementing shard allocation awareness. It allows DGARM to use custom node attributes as awareness attributes to enable Elasticsearch to take your physical hardware configuration into account when allocating shards. If Elasticsearch knows which nodes are on the same physical server, in the same rack, or in the same zone, it can distribute the primary shard and its replica shards to minimize the risk of losing all shard copies in the event of a failure.

## Authentication and RBAC

You can manage and authenticate users natively, or integrate with external user management systems such as LDAP and Active Directory. If none of the built-in realms meet your needs, you can also build your own custom realm and plug it into the Elastic Stack.

At this point there is no directory service identified, DGARM will be leveraging Elasticsearch's Internal authentication methods to manage user authentication.

### User Roles and Permissions

Roles in Elastic are collections of privileges that define the actions a user can perform. Instead of granting privileges directly to users, they are assigned one or more roles that collectively determine their access level.

When a user is assigned multiple roles, the user receives a union of the roles' privileges. This means that assigning additional roles cannot reduce the user's privileges. To adjust a user's access level, you must modify or remove one of their assigned roles.

## Spaces

Kibana Spaces allow the separation of content and dashboards within a single Kibana instance, enabling role-based access control (RBAC) tailored to different user groups or operational personas. Each space can be restricted by role, ensuring that users only access the data and visualizations relevant to their responsibilities.

## Audit Logging

Audit logging is a security feature that allows tracking of access to the Elastic cluster, logging of security-related events. Elasticsearch's audit logging can be used to monitor clusters for suspicious activities, such as unauthorized data access or changes to user security configurations. By enabling and configuring audit logging, DGARM can have visibility into cluster operations and strengthen its overall security posture.

Audit logging is disabled by default in Elasticsearch. To enable it, the following changes need to be made to the settings for Elasticsearch and Kibana in this documentation.

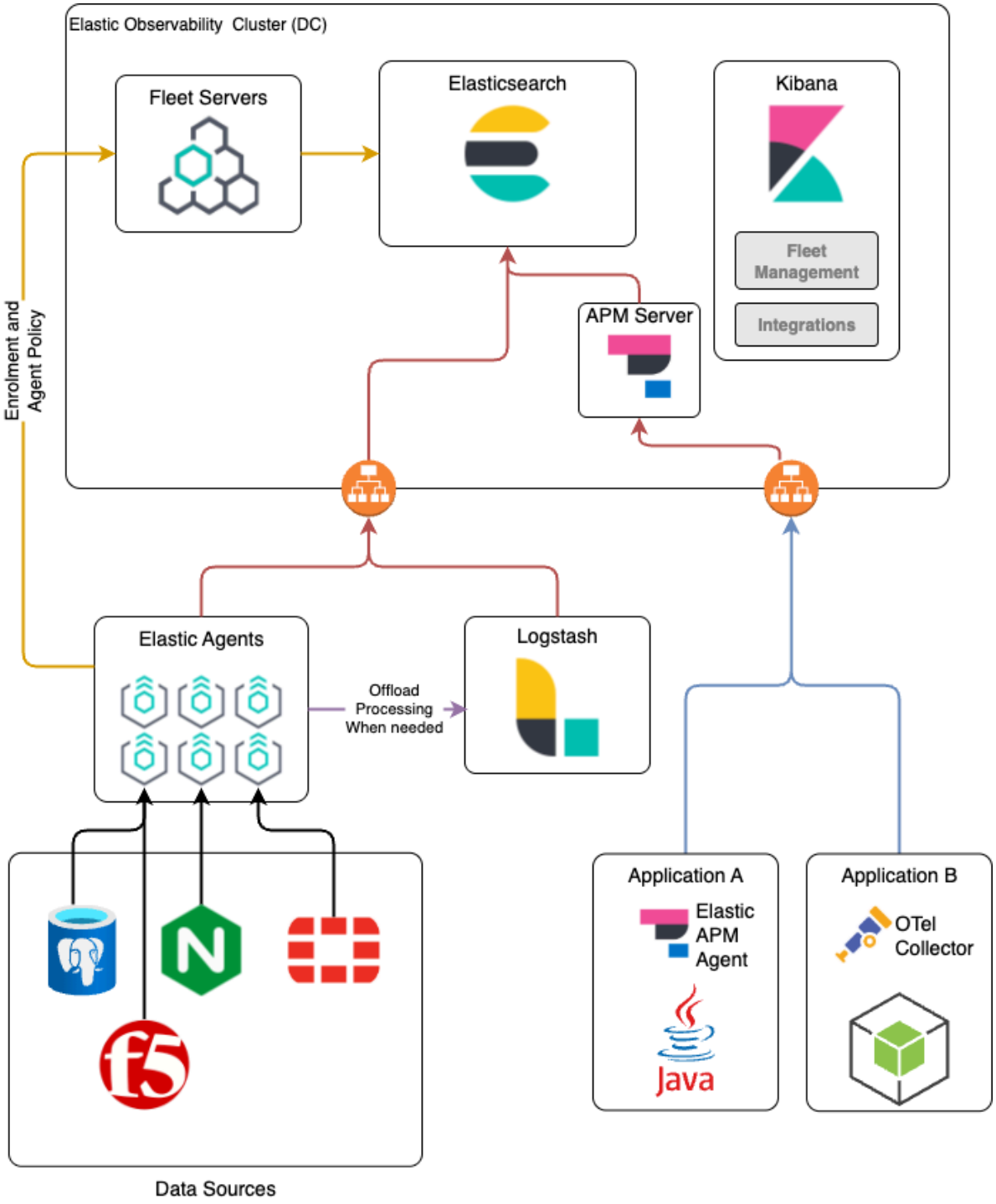
Audit logs can be highly verbose and noisy, especially in active clusters, Elasticsearch events ignore policies and Kibana ignore filters can be used for fine-grained control over which audit events are collected. For more information, see Auditing settings.

As DGARM is starting with a minimal Monitoring cluster, enabling audit logging at this stage may not be optimal. When capacity allows, audit logging can be enabled with appropriate filtering policies to address the required use cases.

# Observability Ingestion Design

## High Level Diagram



## Design Principles

The ingestion architecture for DGARM is designed according to the following ingestion best practices for Elasticsearch:

- All logs ingested will align with the Elastic Common Schema (ECS)
- Logs and metrics will be collected using Fleet-managed Elastic Agent as the default ingestion agent
    - If an Elastic Integrations exists, it will be used to collect the relevant logs and metrics
    - In cases where Elastic Agent is unusable, Beats, Logstash or other mechanisms may be proposed as the fallback option depending on the ingestion scenario.
- Data parsing at Elasticsearch Ingest Pipelines is preferred over Logstash
- All logs and metrics will be encrypted in transit

## Data Sources

As part of the technical discovery phase, The following data sources have been identified for onboarding.

| DataSource | Type | Model | Count | OOTB integration | Custom Integration |
|---|---|---|---|---|---|
| RHEL Server(Virtual) | Server | Red Hat Enterprise Linux 9.5 (Plow) | 800/900 | System Logs & Metrics | |
| Server | Server | HPE ProLiant DL380 Gen11 & HPE ProLiant DL385 Gen11 | 113 | System Logs & Metrics | |
| Kubernetes | | Openshift | | Kubernetes | |
| emudhra | Application | securepass (iDAM) | NA | | NA |
| postgres | Database | PostgreSQL 16.9 | NA | PostgreSQL | |
| Data Fabric HP Ezmeral | Platform/DB | 7.8.0 | NA | | Custom |
| Unified Analytics HP Ezmeral | Platform/DB | 1.5.x | NA | | Custom |
| IBM mq | Application | 9.4.0.10 | NA | IBM MQ | |
| Neo 4J | Application | jws-6.1 | NA | | |
| Nginix+API Gateway | Application | F5 nginx/1.27.4 | NA | | Custom |
| Jboss Web server | Server | jws-6.1 | NA | | Custom |
| Git Hub | Application | | NA | GitHub | |

| | | | | | |
|---|---|---|---|---|---|
| Manage Engine -ITSM | Application | | NA | | Custom / Syslog |
| Manage Engine - Patch | Application | | NA | | Custom / Syslog |
| Tenable | Application | | NA | Tenable | |
| Imperva | Application | | NA | Impreva | |
| Cisco SAN Switch/Storage | Storage | Cisco | 6 | Cisco Nexus | Syslog / SNMP |
| NetApp Storage | Storage | Netapp | 6 | | NetApp Metric API or Netapp Harvester/Prometheus Input integration |
| Veritas Back up | Backup | Veritas | 4 | | Custom / API |
| Palo Alto Firewall - Internal | Security | PA-3410 | 4 | Palo Alto Next-Gen Firewall | |
| Fortinet Firewall - External | Security | Fortigate FG-401F | 4 | Fortinet FortiGate | |
| Sandbox | Security | Fortigate FSA-500G | 4 | Fortinet FortiGate Firewall Logs | |
| Juniper Router | Network | Juniper SRX 2300, SRX380 | 14 | Juniper SRX | |
| Switches | Network | Aruba Networking CX 8100, Core - HPE Aruba 8325-32C, OOB - Aruba Networking CX 6200F | 36 | HPE Aruba CX | |
| WAF | Security | F5 WAF -R2600 | 4 | F5 BIG-IP | |
| DDOS | Security | F5 -DDOS - R2600 | 4 | F5 BIG-IP | |
| Proxy | Security | Forcepoint | 4 | Forcepoint Web Security | |
| DLP | Security | Forcepoint | Virtual | Forcepoint Web Security | |
| F5 Load balancer | Security | F5 | Virtual | F5 BIG-IP | |

Application Performance Monitoring data sources (Java and Node.js.)

| Application/Microservice Name | Type |
|---|---|
| flightScheduleService | Microservice |
| watchlistService | Microservice |

elastic

| | |
|---|---|
| riskManagement | Microservice |
| mapService | Microservice |
| manToMachineService | Microservice |
| userManagementService | Microservice |
| masterdataservice | Microservice |
| regAuthService | Microservice |
| searchService | Microservice |
| auditLoggingService | Microservice |
| communicationService | Microservice |
| documentService | Microservice |
| adminweb | Microservice |
| cacheManagementService | Microservice |
| edifactProcessingService(MQ Consumer) | Microservice |

# Fleet and Elastic Agent

## Elastic Agent

Elastic Agent will serve as the primary ingestion service to collect logs and metrics from the various mentioned data sources. Elastic agents will be deployed at scale, and centrally managed through Fleet Server. The Elastic Integration repository offers a large number of pre-configured and tailored options for readily integrating with common technologies and data sources to easily collect log and metrics information from applications.

## Fleet Server

Fleet Server a special Elastic Agent which connects other Elastic Agents to Fleet for agent management. It acts as a control plane for updating agent policies, collecting status information and coordinating actions across Elastic Agents.

## Air Gapped

In case the deployments do not have internet access. The Elastic Package Repository (EPR) and the Elastic Artifacts Repository needs to be self-hosted. The EPR hosts the elastic integrations which contain the out of the box log parsers, dashboards and other content. The Elastic Artifacts Repository

hosts the binaries which are required to install and upgrade the Elastic Agents. Instructions to self-host EPR can be found here. Instructions to self-host artifacts repository can be found here.

### High Availability

There will be two Fleet Servers deployed in the observability cluster. These Fleet Servers will be spread across two VMs.

Fleet Server is stateless, therefore no local configuration to be maintained. Fleet servers can be placed behind a load balancer or multiple Fleet Server URLs can be specified to provide automatic failover.

## Scaling

Fleet servers sizing will depend on the number of agents managed. The following documentation outlines the hardware requirements per agent count. The sizing provided in the cluster topologies will be sufficient for up to 3000-5000 agents.

# Logstash

Although not the primary ingestion tool, Logstash is included in the architecture to alleviate potential bottlenecks which may occur as the deployment scales to larger volumes. There are also some specific use cases which require Logstash. Some potential use cases for Logstash are listed below:

1. Offload parsing, enrichment and extract, transform and load (ETL) functionality off the Elasticsearch cluster for high throughput or compute heavy workloads.
2. Serve as a network and log aggregation point for Elastic Agents to:
    a. Reduce the number of concurrent connections into Elasticsearch
    b. Simplify networking configuration, including firewall rules
3. Perform further processing of APM data
4. Perform specific log actions such as splitting events and aggregating events

## High Availability

Logstash can be deployed across multiple VMs as we have only one availability zone. A network load balancer will be deployed in front of Logstash to spread the connections across multiple instances.

This load balancer will be TLS passthrough to ensure that TLS verification can happen between the Logstash and Elastic Agent to ensure log traffic is encrypted. Currently, the only planned input to Logstash is from Elastic Agent on TCP/5044.

## Queuing

By default Logstash utilises an in-memory queue to buffer events during processing. It is possible to configure a persistent queue which will write events to disk. This protects against message loss if Logstash is terminated unexpectedly and the in-flight events have not been fully transmitted. The persistent queue also allows Logstash to absorb bursts of events without an external buffering technology such as Kafka. However, as the events are committed to disk prior to sending to the configured output, this will introduce latency and additional performance overhead.

## Scaling

Logstash should be scaled horizontally to optimise the hardware utilisation. Ideally, Logstash hosts do not need to exceed 16GB of RAM (8GB of JVM Heap). When monitoring Logstash, pay close attention to CPU since it is usually the limiting factor for maximum throughput. Heap usage, and disk performance if persistent queues are enabled, are also key metrics. Logstash metrics can be collected and sent to the monitoring cluster and analysed to make scaling decisions. The following metrics are important to consider:

1. CPU Utilisation
2. JVM usage
3. Pipeline Events metrics
4. Pipeline Flow metrics
5. Persistent Queue metrics
6. Disk metrics - collected by Elastic Agent or Metricbeat system module

# APM Server

Elastic APM server will collect logs, metrics and traces sent through APM agents. It transforms these into documents which can be sent directly to Elasticsearch or to Logstash for further processing. Elastic APM server also supports APM data coming from OpenTelemetry and will enable the use of OpenTelemetry traces to be used with Elastic Observability.

## High Availability

APM server will be deployed across two VMs. The endpoints will be exposed via the load balancer. This load balancer will be TLS passthrough to ensure that TLS verification can happen between the APM servers and APM sources.

## Scaling

APM Server performance can be dependent on many factors, primarily the number of transactions per second and the number of APM agents/sources sending data. The following documentation describes some of the scaling guidance which Elastic has recommended for APM Server. APM server is a stateless ingest component, therefore it can be scaled horizontally with ease.

## Sampling Strategies

Sampling controls how much trace data is collected and stored, balancing cost and visibility. There are 3 sampling strategies: no sampling, head-based sampling and tail based sampling.

### Definition

**No Sampling**

- All traces are captured and ingested.
- Provides complete visibility but may generate a high volume of data.
- Suitable for low-traffic apps or short-term troubleshooting.

**Head-Based Sampling**

- Sampling decision is made at the start of the request.
- Simple and low-overhead.
- Limitation: rare or slow transactions may be missed.

### Tail-Based Sampling

- Sampling decision is made after the request completes, based on content (e.g., errors, latency).
- Allows intelligent selection of important or anomalous traces.
- Requires Elastic APM Server as only Elastic APM Server currently supports tail-based sampling.

#### Comparison of Sampling Strategies

The table below provides comparisons for the 3 sampling strategies

| Criteria | No Sampling | Head-Based Sampling | Tail-Based Sampling |
|---|---|---|---|
| **Sampling Decision** | All traces are retained | At the start of a request | After the request completes, based on outcome |
| **Setup Complexity** | Simplest | Simple, fully OTEL-native | More complex – requires Elastic APM Server |
| **Latency Impact** | None | Minimal | Slightly higher due to span buffering |
| **Trace Completeness** | Full visibility of all transactions | May miss rare or low-frequency traces | High accuracy in capturing rare or anomalous traces |
| **Data Volume & Storage Cost** | High | Moderate (controlled by sampling ratio) | Could be lower (selective trace retention) |
| **Flexibility** | Limited control over data volume | Flexible, tunable via OTEL sampling configs | Sampling policy handled by APM Server logic |

## Log and Metrics Collection

Elastic Agents will be deployed on endpoints located within both on-premises infrastructure and virtual machines, and where required. These agents will be responsible for collecting logs and metrics from distributed systems and workloads, ensuring consistent and centralized data ingestion across the entire ecosystem.

## Elastic Integrations

Elastic Integrations are Elastic prebuilt packs which reduce the time taken to onboard and derive value from log sources. Each integration comes pre-packaged with assets that support your needs and allow you to easily collect, store, and visualize any data from any source. which may be relevant to DGARM's observability use cases.

Many of the log sources DGARM wishes to onboard already have an Elastic integration available, see the table above. Log sources onboarded with an integration will be automatically parsed and mapped into the Elastic Common Schema (ECS). Log parsing is performed through Elasticsearch ingest pipelines which run on the Ingest Nodes.

## Custom Data Source Integration

### Log Collection - HTTP endpoint

In scenarios where external systems can push events to a HTTP Endpoint, the custom HTTP Endpoint Log integration initializes a listening HTTP server that collects incoming HTTP POST requests containing a JSON body. The body must be either an object or an array of objects. Any other data types will result in an HTTP 400 (Bad Request) response. For arrays, one document is created for each object in the array.

### Log Collection - Syslog

If logs are forwarded using syslog, then Elastic Agents will be set up as syslog listeners behind a network load balancer. These Elastic Agents will run either an existing integration or the custom TCP/UDP input.

To simplify the parsing process, each different syslog log type or format should be forwarded to a unique port. E.g. syslog from UNIX servers should go to port 5514, syslog from a network device can go to port 5515

### Log Collection - API

In some cases an API will need to be polled to retrieve data. In this case, if an integration exists, it will be utilised otherwise Elastic Agent Custom API integration or the Custom CEL integration will be used to collect. Note that proxy access might need to be provided for these API endpoints.

### Metrics Collection

The collection method for metrics will be determined by the Elastic integration. If no integration exists, one of the following options can be used:

1. If the application has APM enabled, some metrics will be collected through the APM agents.
2. If an metrics API exists, the metrics can be polled using the Elastic Agent Custom API input
3. If metrics are forwarded using syslog, Elastic Agents can be set up as syslog listeners behind a network load balancer. These Elastic Agents will run either an existing integration or the custom TCP/UDP input.
4. In cases where external systems can push metrics to a HTTP Endpoint, the custom HTTP Endpoint Log integration initializes a listening HTTP server that collects incoming HTTP POST requests.

## Traces Collection - Elastic APM

Elastic APM will be used to collect logs, metrics and traces from applications. Generating application traces and metrics generally requires the direct instrumentation of an APM Agent into the application code. The instrumentation process is different for each language and requires a separate APM Agent. Java and Node.js. are the common application languages used in DGARM applications.

The APM data can be tagged with metadata and resource attributes to correlate traces with application logs and metrics. Elastic provides APM agents for a variety of languages and will tag the APM data automatically. The instrumentation process depends heavily on the language.

Refer to respective agent's documentation for more details on supported instrumentation approaches:

- Java agent
- Node.js agent

## Traces Collection - OpenTelemetry

Traces generated by OpenTelemetry can also be collected via Elastic APM Server through the OTLP protocol. There are two patterns to receive data from OpenTelemetry:

1. Traces, metrics and logs forwarded to APM Server from an OpenTelemetry Collector
2. Traces metrics and logs delivered directly to APM Server from the OpenTelemetry agent.

When exporting from an OpenTelemetry Collector, it is important to choose the `oltp` exporter, as this sends data in the correct format to APM Server which will allow the data to be formatted correctly for use within the Kibana Observability applications.

## Real User Monitoring (RUM)

The frontend application plays an important role in user experience. To gain visibility into client-side performance, this application is instrumented using the Elastic Real User Monitoring (RUM) JavaScript Agent.

**APM RUM JavaScript agent -> APM Server -> Elasticsearch**

In this architecture:

- The Elastic JavaScript Agent is embedded into the frontend codebase. Once loaded in the browser, it captures a variety of performance metrics, including page load times, route changes, JavaScript errors, and HTTP requests made from the client.
- The agent also collects user context such as browser metadata, geolocation (if enabled), session ID, and user agent string. This information is essential for analysing frontend performance in real-world user conditions.
- All telemetry data is sent directly from the browser to the Elastic APM Server, which processes and forwards it to Elasticsearch for storage and analysis.

# Data Streams

A data stream lets you store append-only time series data across multiple indices while giving you a single named resource for requests. Data streams are well-suited for logs, events, metrics, and other continuously generated data. There are multiple benefits to utilising data streams:

- Automatic index management using Index Lifecycle Management (ILM)
- Automatic index bootstrapping for index rollover
- Automatic index rollover to ensure optimal shard sizes
- Elastic Integrations optimise storage and mappings when using the recommended data stream naming conventions
- Leverage Elastic pre-built content by following data stream naming conventions
- Leverage storage efficiency features such as LogsDB

## Naming Convention

All logs should be written to data streams. Data streams are designed specifically for append-only data. Data streams must follow the recommended Elastic naming conventions i.e. `<type>-<dataset>-<namespace>`. The `type` and `dataset` is typically static and based on the type of data being ingested. Below are samples based on some custom data sources:

- Netapp Metrics
    - metrics-netapp.status-default
- Veritas Custom Logs
    - logs-veritas.eventlog-default
    - logs-veritas.admin_jobs-default
    - logs-veritas.alerts-default

# Sharding Strategy

Each index in Elasticsearch is divided into one or more shards. Each document in an index belongs to a shard. Each shard has some overhead and having too many shards can negatively impact

Elasticsearch performance and cluster stability. Therefore, enforcing a strict sharding strategy is crucial. The following guidelines should be followed for shard sizing:

1. Shards should be between 10GB and 50GB in size.
2. In cases where logsdb data stream type is used, 10-30GB is an ideal size.
3. Shards should contain no more than 200 million documents.
4. No more than 3000 indices per 1 GB of master heap.

Cluster shard limits prevent creation of more than 1000 non-frozen shards per node, and 3000 frozen shards per dedicated frozen node. Make sure you have enough nodes of each type in your cluster to handle the number of shards you need.

## Index Lifecycle Management (ILM)

ILM automatically transitions time series datasets to different data tiers over time. This allows cheaper hardware to be leveraged for data which is older. This optimises the hardware utilisation and saves infrastructure costs.

Arbitrarily assigning ILM phases and retention durations which apply broadly across all data will lead to a suboptimal cluster configuration and over utilisation of storage. Retention strategies should be defined based on the usage requirements of each data source and the strengths of each tier. Consider the following for each tier:

- Hot Tier - Primarily used for Ingestion, near real-time search for alerting, live dashboards
- Warm Tier - Primarily used for low latency search and for processing large data sets and reports

Initially DGARM will utilise the following data retention:

- 3 days Hot
- 27 days Warm

It is recommended to define separate ILM policies for logs and metrics, as they may have different retention requirements.

Usage should be monitored and the ILM policy should be tuned to match the appropriate usage patterns.

To ensure optimal shard sizing, the rollover phase of the ILM will be configured with the following parameters:

- Max primary shard size - 50GB
- Maximum age - 30 days or min shard size (to prevent too many small shards from being generated)
- Maximum documents per shard - 200 million

The examples below illustrate how logs and metrics can be managed with different lifecycle requirements. These ensure data is retained for the appropriate number of days and that rollover is based on shard best practices.

```
None
# Sample ILM policy for logs
PUT _ilm/policy/DGARM-logs-3h-27w-d
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_age": "30d",
            "max_primary_shard_size": "50gb"
          },
          "set_priority": {
            "priority": 100
          }
        },
        "min_age": "0ms"
      },
      "warm": {
        "min_age": "3d",
```

```
        "actions": {
          "set_priority": {
            "priority": 50
          }
        }
      },
      "delete": {
        "min_age": "30d",
        "actions": {
          "delete": {
            "delete_searchable_snapshot": false
          }
        }
      }
    }
  }
}

# Sample ILM policy for metrics
#PUT _ilm/policy/DGARM-metrics-3h-27w-d
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_age": "30d",
            "max_primary_shard_size": "50gb"
          },
          "set_priority": {
            "priority": 100
          }
        },
        "min_age": "0ms"
      },
```

```
      "warm": {
        "min_age": "3d",
        "actions": {
          "set_priority": {
            "priority": 50
          }
        }
      },
      "delete": {
        "min_age": "30d",
        "actions": {
          "delete": {
            "delete_searchable_snapshot": false
          }
        }
      }
    }
  }
}
```

## Snapshots

A snapshot in Elasticsearch is a backup of a running cluster's data and state. It captures a point-in-time copy of indices and optionally the cluster metadata, and stores it in a registered snapshot repository.

Snapshots are stored incrementally and multiple snapshots can reference the same underlying objects in a snapshot repository. Therefore, multiple snapshots covering the same indices do not increase the storage requirements in a Blob Storage provided that the snapshots reference the same repository. Deleting a snapshot will not remove the data in the repository as long as there is at least

one snapshot still attached. This is important as it means that searchable snapshots can be deleted without impacting the backups and vice versa.

## Repository

To enhance redundancy and ensure business continuity, it is recommended to configure an off-site repository (like Azure, AWS, GCP, Shared file system) for Elasticsearch snapshots. In the case of DGARM, snapshots will be stored on a shared file system repository backed by a Network-Attached Storage (NAS) solution. This provides fault isolation by storing backups outside the primary cluster storage, ensuring they remain accessible even in the event of a site-wide or hardware failure. Once the NAS is mounted and accessible by all eligible master and data nodes, the snapshot repository can be registered using the `fs` (file system) type. This setup aligns with Elastic's best practices for resilient backup strategies, offering a scalable, shared, and cost-effective solution suitable for high-frequency snapshot operations and long-term retention..

## Snapshot Lifecycle Management (SLM)

Cluster snapshots can be managed using Snapshot Lifecycle Management (SLM). Note this is different from the searchable snapshots which are managed as part of ILM actions.

When configuring long term snapshot retention, it is important to consider the frequency of the snapshots as well as the retention of a snapshot. Each snapshot has some overhead on the master nodes, so having an appropriate snapshot lifecycle management strategy is important for cluster stability.

The recommended strategy is to apply multiple SLM policies with varying intervals and retention to provide flexibility in both snapshot granularity as well as retention length. The following three policies could be used in conjunction:

1. Hourly snapshots which are retained for 24 hours - 24 total snapshots
2. Daily snapshots which are retained for a month - 31 total snapshots
3. Monthly snapshots which are retained for a year - 12 total snapshots

Combining the above three policies will ensure that snapshots are taken every hour minimising the potential data loss and/or re-ingestion required if recent data needs to be restored. In addition, it reduces the overall number of snapshots required by taking snapshots at longer intervals. This process is described in detail in the following documentation.

# Alerting

## Overview

Alerting enables you to define rules, which detect complex conditions within different Kibana apps and trigger actions when those conditions are met. Alerting is integrated with Observability, Security and Machine Learning. It can be centrally managed from Stack Management and provides a set of built-in connectors and rules for you to use.

## Alerting Use Cases

During the engagement, potential alerting use cases were discussed to support proactive issue detection and resolution. Examples included the CPU Usage Alert, monitoring system performance by triggering notifications when thresholds are exceeded, and the Error Logs Alert, identifying error-level events in ingested data for early issue detection.

Elastic's built-in connectors, such as Email (SMTP), Slack, Microsoft Teams, ServiceNow, and custom webhooks, offer flexible options for real-time alert delivery. DGARM team will configure preferred connectors, such as email for direct alerts, Microsoft Team, or Zendesk for collaborative notifications as needed. Leveraging these capabilities will ensure timely delivery of alerts and fully utilize Elastic's monitoring and alerting potential.

# Machine Learning and AI

Elastic's Machine Learning (ML) and Anomaly Detection features enable proactive monitoring by identifying unusual patterns in logs, metrics, and time-series data. These tools automatically detect anomalies such as spikes in log rate, error log surges, or unexpected network activity, helping DGARM address issues before they escalate. Prebuilt ML jobs introduced during the engagement,

such as log rate analysis and infrastructure monitoring, can be customized and integrated with alerting rules to ensure timely notifications for critical anomalies.

The AI Assistant simplifies data exploration with natural language queries, making it easier for teams to interact with Elastic data without requiring advanced query skills. It provides context for detected anomalies, actionable insights for resolution, and proactive recommendations to optimize performance. By combining ML-powered anomaly detection with the AI Assistant, DGARM can enhance its observability and security efforts, enabling faster response times and more efficient operations.

# Search Design

Elastic has reviewed the requirements and done a brief analysis of the functional requirements, shared in *SearchFRS.docx* on the 26th of August. An attempt has been made to identify key areas of Elasticsearch implementation and assist in the design of those. The configurations and artefacts are provided as guidance and are not complete, final nor ready for production use.

## Requirement Analysis

The requirements include the following identified key elements:

1. The use case is search, i.e. users may search across all records
2. The search indices must support all CRUD operations
3. Records are time based, i.e. they will receive CRUD for a limited period of time and then become static
4. Record retention polices control how long records are kept
5. Search features are provided in a structured form and include basic filtering, auto complete and fuzzy search

### Recommended Change - Filtering on Number of Arrivals / Departures

Elastic has specifically noted the *Passenger Search - Journey Details* requirements:

- 2. g. Number of Arrivals >=
- 2.h. Number of Departures >=

As per the functional requirements the filters should take into account the time filters applied by the user. Such filtering on aggregated data is not practical in Elasticsearch. Elastic recommends adding a metric that is easily searchable, e.g. number of trips last six months and that can be statically assigned to the passenger as part of the passenger record.

## Ingestion

Data will be ingested using custom Python code running within the Iceberg platform. Design of the application is out of scope for this document, Elastic makes the following overall recommendations:

1. Utilise the official Elasticsearch python client. This will speed up development and reduce risk
2. Use the Bulk API to write data to elasticsearch, single write requests are very expensive at scale. The Python client comes with helper classes for this purpose.
3. Size bulk requests appropriately, aiming for a 1-10MB payload is a good starting point
4. Use multiple workers to send data to Elasticsearch, however ensure that indexing load does not harm search speed
5. See also tune for disk usage, note don't force merge indexing until after all writes have completed for an index

## Index Management

Since Elasticsearch will need to support all CRUD operations, Elastic recommends using indices named with a date suffix.

### Index Naming

Each index should be named according to the convention <data set name>-<date suffix>. Elastic recommends using monthly indices, as a suggestion the below example names can be used:

- flights-2025-08
- passenger-2025-08
- crew-2025-08

When writing data to Elasticsearch, the destination index suffix should be based on a date field that does not change for a specific record, for example, scheduled *departure date*.

## Index Retention

Index retention should be managed by using Elasticsearch index lifecycle management (ILM). Elastic recommends using the following policy for a 6 month policy:

1. Hot
    a. Rollover disabled
    b. Set index priority 100
2. Delete
    a. When 217 days old*

* Note we need to keep the index for seven months, since the time is calculated from index creation and each index contains data up to creation date + one month.

The ILM policy can be created using the following API call:

```
None
PUT _ilm/policy/dgarm
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "set_priority": {
            "priority": 100
          }
        }
      },
      "delete": {
        "min_age": "217d",
        "actions": {
          "delete": {
```

```
            "delete_searchable_snapshot": true
          }
        }
      }
    }
  }
}
```

## Shard Sizing

- Elastic recommends 1 primary and 1 replica shard for each index as a starting point.
- If the monthly primary shard size grows beyond 50GB, Elastic recommends adjusting the primary shard setting to a sufficient number to bring down the shard size under 50GB.
- If the monthly index size, counting primaries only, grows beyond 200GB for any dataset, Elastic recommends transitioning to daily indices and setting the primary shard back to 1.

## Index Template

Elastic recommends using index templates to apply mappings and settings to indices. Component templates can be used to reuse configuration between different index patterns. See further down for an example index template.

# Autocomplete

The search application for searching fields requires autocomplete on the following fields:

1. Name
2. Airport
3. Nationality
4. Operator

## Simple Autocompletion

For simple strings, i.e., 2-4 in the list above, elasticsearch recommends mapping each field as *keyword* and using the terms enum API for autocompletion. The same approach can be applied to other fields, including pnr, document number, origins and destination.

## Name Autocompletion - Option 1 (recommended)

The user may be typing  names starting with either the first name or the last name. Elastic recommends introducing two fields;

1. firstname_lastname
2. lastname_firstname

When autocompleting on name, the search client sends two terms enum requests in parallel,one to each field, and merges the results. This is following the same approach as the simple autocompletion. The selected option can then be used to make an exact query against the appropriate keyword field, or a text query as desired.

## Name Autocompletion - Option 2

If DGARM requires autocomplete to combine advanced query features such as fuzzy search, with autocomplete, the following approach can be used.

1. Copy the full name into two separate fields, for example:
    a. fullname_complete, mapped as search-as-you-type
    b. fullname, mapped as keyword
2. Generate name suggestions using a modified search-as-you-type query,, see example below
    a. Optional: apply any other user selected filters to discard irrelevant suggestions
3. Use aggregations to remove duplicate names

**Note** Elastic does not recommend this option due to search latency and the impact on user experience. If DGARM requires this option, please consider:

1. Benchmark performance, for example using ESRally

2. Ensure there is a delay between keypresses and sending an autocomplete request, so that not every keypress causes a request
3. Apply user selected filters to reduce the result set
4. Consider increasing the CPU count of Elasticsearch nodes if performance suffers

None

```
GET passengers/_search
{
  "query": {
    "size": 0,
    "bool": {
      "must": [
        {
          "multi_match": {
            "query": "mattias b",
            "type": "bool_prefix",
            "fields": [
              "fullname_complete",
              "fullname_complete._2gram",
              "fullname_complete._3gram"
            ]
          }
        },
        {
          "match": {
            "name_complete": {
              "query": "fullname_complete",
              "fuzziness": "AUTO"
            }
          }
        }
      ]
    }
  },
  "aggs": {
```

```
    "sample": {
      "sampler": {
        "shard_size": 100
      },
      "aggs": {
        "name": {
          "terms": {
            "field": "fullname"
          }
        }
      }
    }
  }
}
```

Note that the sampler aggregation is used to reduce the number of results analysed to the top N relevant hits in each shard, in this case 100.

## Ingest Pipeline

To populate *firstname_lastname* and *lastname_firstname* fields, an index pipeline can be used. The pipeline below uses the set processor. The pipeline can be automatically executed by adding the *default_pipeline* setting to each index.

```
None
PUT _ingest/pipeline/passenger
{
  "processors": [
    {
      "set": {
        "field": "firstname_lastname",
        "value": "{{{firstname}}} {{{lastname}}}"
```

```
      }
    },
    {
      "set": {
        "field": "lastname_firstname",
        "value": "{{{lastname}}} {{{firstname}}}"
      }
    }
  ]
}
```

## Index Template

Please see below for an example index template, including mappings. Note that fields have generally been mapped as keyword to be used as filters, either through the use of drop downs or autocomplete, see the *Autocomple* section.

None

```
PUT _index_template/passenger
{
  "index_patterns": [
    "passenger-*"
  ],
  "template": {
    "settings": {
      "index.lifecycle.name": "dgarm",
      "number_of_shards": 1,
      "number_of_replicas": 1
    },
    "mappings": {
      "properties": {
        "pnr": {
```

```
      "type": "keyword",
      "normalizer": "lowercase"
    },
    "first_name": {
      "type": "keyword",
      "normalizer": "lowercase"
    },
    "last_name": {
      "type": "keyword",
      "normalizer": "lowercase"
    },
    "firstname_lastname": {
      "type": "keyword",
      "fields": {
        "text": {
          "type": "text"
        }
      }
    },
    "lastname_firstname": {
      "type": "keyword"
    },
    "date_of_birth": {
      "type": "date"
    },
    "nationality": {
      "type": "keyword"
    },
    "document_number": {
      "type": "keyword"
    },
    "document_expiry_date": {
      "type": "date"
    },
    "gender": {
```

```
        "type": "keyword"
      },
      "contact_number": {
        "type": "keyword"
      },
      "contact_email": {
        "type": "keyword"
      },
      "travel_agency": {
        "type": "keyword"
      },
      "cabin_class": {
        "type": "keyword"
      },
      "origin": {
        "type": "keyword"
      },
      "destination": {
        "type": "keyword"
      },
      "scheduled_departure": {
        "type": "date"
      },
      "scheduled_arrival": {
        "type": "date"
      },
      "actual_departure": {
        "type": "date"
      },
      "actual_arrival": {
        "type": "date"
      }
    }
  }
}
```

```
}
```