THALES GCP GENAI/ML RESOURCE KIT

PROTECTING SENSITIVE DATA

THALES

Contents

1 | Introduction.......... 2

2 | Sourcing Phase. 4

3 | Training Phase 7

4 | Deployment & Use Phase.......... 10

5 | Appendix. 14

Bring Your Own Keys and Hold Your Own Keys 14

Bring Your Own Encryption (BYOE).......... 16

Finding sensitive data sources and protecting it. 17

Examples 20

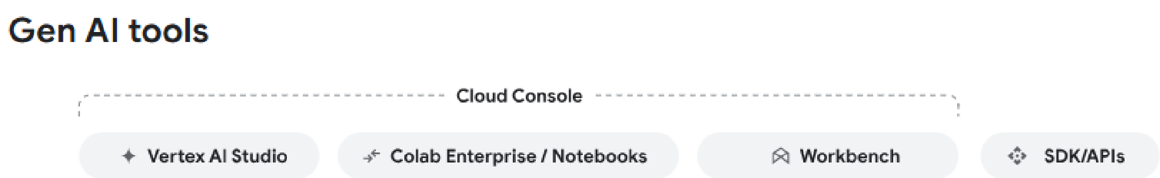
1 | Introduction

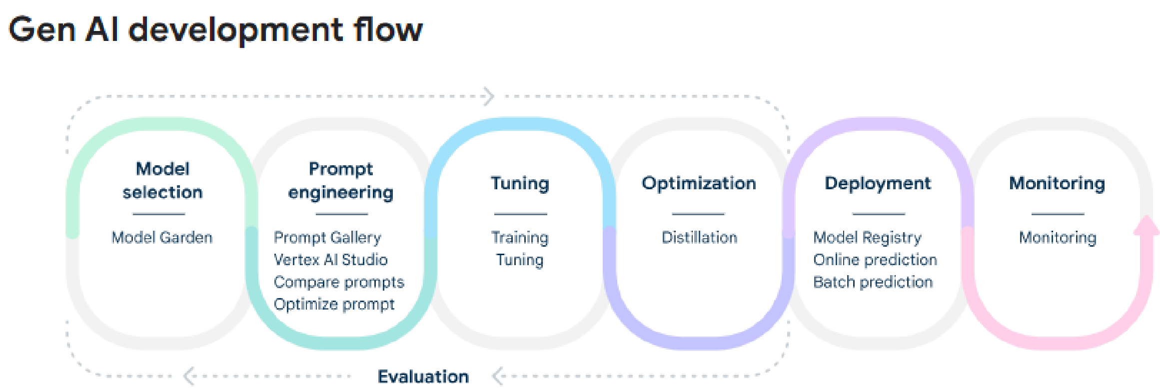
Generative AI's widespread appeal stems from its potential to transform industries. Businesses aim to leverage it for better customer experiences, streamlined operations, and faster processes. However, the success of these AI/ML applications hinges on data and its context, making the understanding and protection of sensitive enterprise data essential for proper and secure deployment.

Tuning or extending generative AI for specific business uses necessitates data, similar to other AI/ML workloads. A major concern for organizations is minimizing the risk of using their own data, which may contain sensitive personal information (PI/PII), for training. This personal data is often embedded within necessary contextual information for the model to function effectively.

From a security perspective there are many potential security risks associated with the implementation of GenAi systems such as Data Poisoning Definition, Backdoor insertion Definition ,Model theft definition and Prompt injection definition attacks. ***The focus of this document will be on implementing Thales solutions with GCP capabilities to ensure sensitive data such as PII, PCI, etc will be protected.***

Listed below is a diagram highlighting the GCP GenAI tools and development workflow.



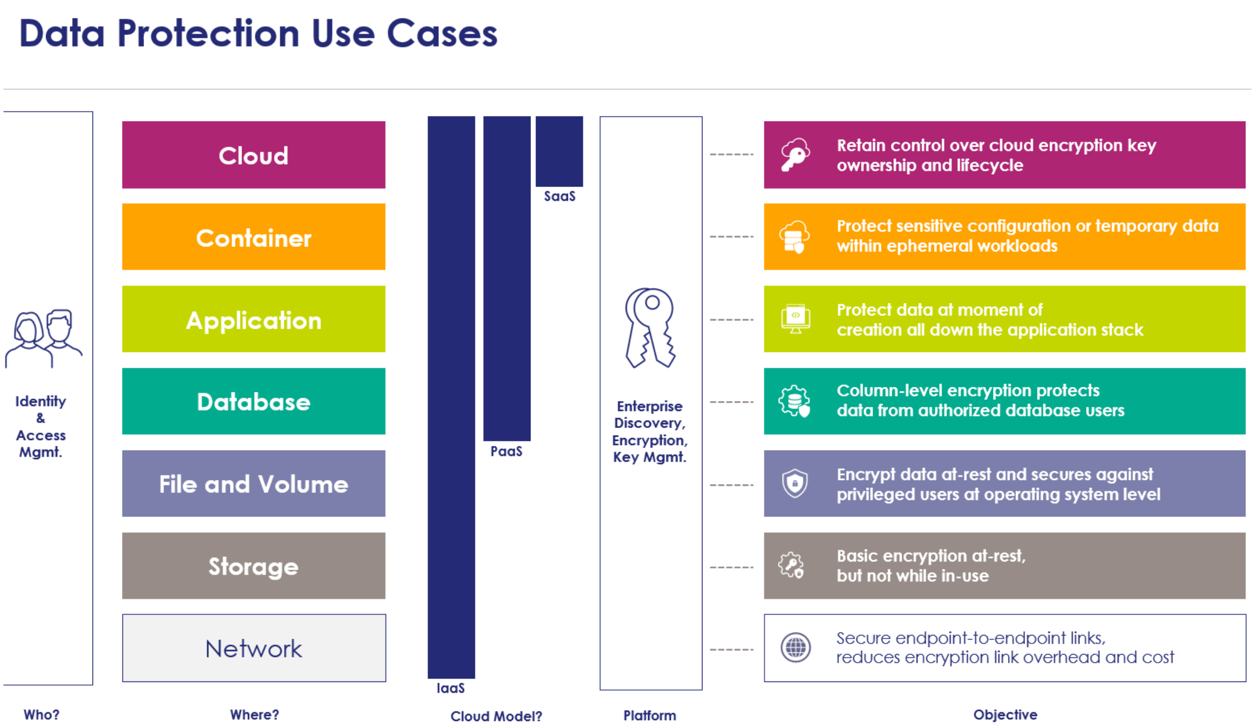


During tunning phase the GCP bucket is used to load training data which incorporates the GCP KeyRing for encryption keys to protect the data.

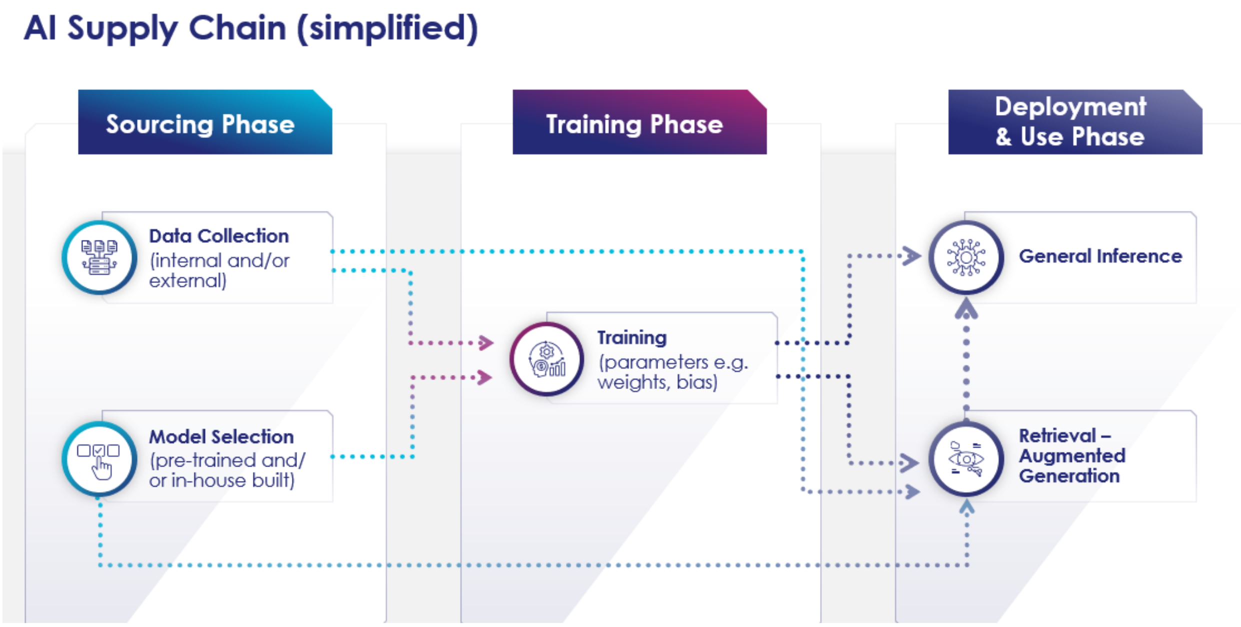
Thales provides customers the ability to populate the GCP KeyRing with key material created in an external key manager (CipherTrust Manager) using customer managed keys to achieve better control and compliance. See Appendix on Bring Your Own Encryption & Hold Your Own Key for more information.

One thing to keep in mind is that using a strategy of Bring Your Own Key (BYOK) or Hold Your Own Key (HYOK) may be sufficient for many use cases, depending on the classification of your data, it may not be appropriate for use cases where the data is highly sensitive. When implementing GCP native encryption, it is important to know what threats you are protected from with this kind of strategy. The diagram below highlights the different type of encryption and the threat you are protected from. GCP "Native" encryption is somewhat similar to Full Disk Encryption (FDE) that customers deploy for their on-premises disk. The difference with GCP KeyRing is that instead of having one key for the entire disk GCP provides customers the ability to have their own key for just their "service or portion of the disk" (whatever the key is protecting in the CSP). The other difference is that customers have the ability to disable the key to prevent access to the service which makes the entire service unavailable. It is important to keep in mind ***you are NOT protected once the disk is spinning and a bad actor logs onto the data center to infiltrate your systems and data***.

As seen from the diagram above when using the GCP GenAI capabilities some of the locations of data reside on S3 buckets in the customer's control. Looking at the encryption stack below it is necessary to deploy a different kind of encryption, bring your own encryption (BYOE) to prevent this kind of threat. For more information on how this can be accomplished see the section titled Bring Your Own Encryption in the Appendix.



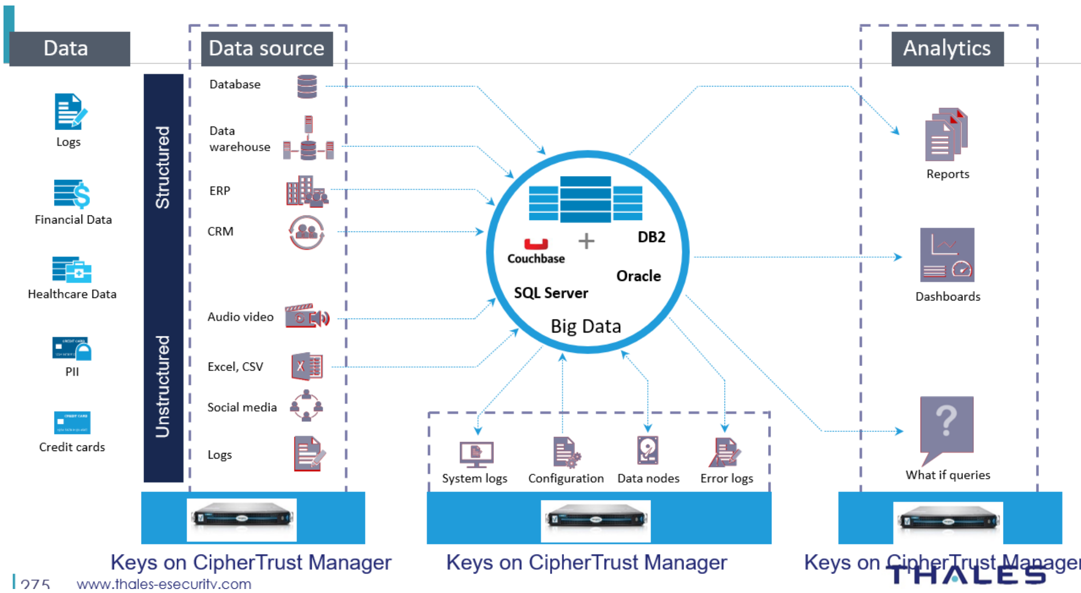
The Generative AI workflow can be broadly categorized into three distinct phases. First, the **sourcing phase** involves critical decisions regarding data collection, encompassing the gathering and preparation of relevant datasets, and model selection, where the appropriate pre-trained or custom model is chosen based on the desired application. Next, the **training phase** focuses on fine-tuning the selected model using the sourced data, optimizing its parameters to generate desired outputs. Finally, the **deployment phase** entails making the trained model accessible for practical use, encompassing both general inference, where the model generates outputs based on user prompts, and Retrieval Augmented Generation (RAG), which enhances responses by grounding them in external knowledge sources for improved accuracy and contextuality.



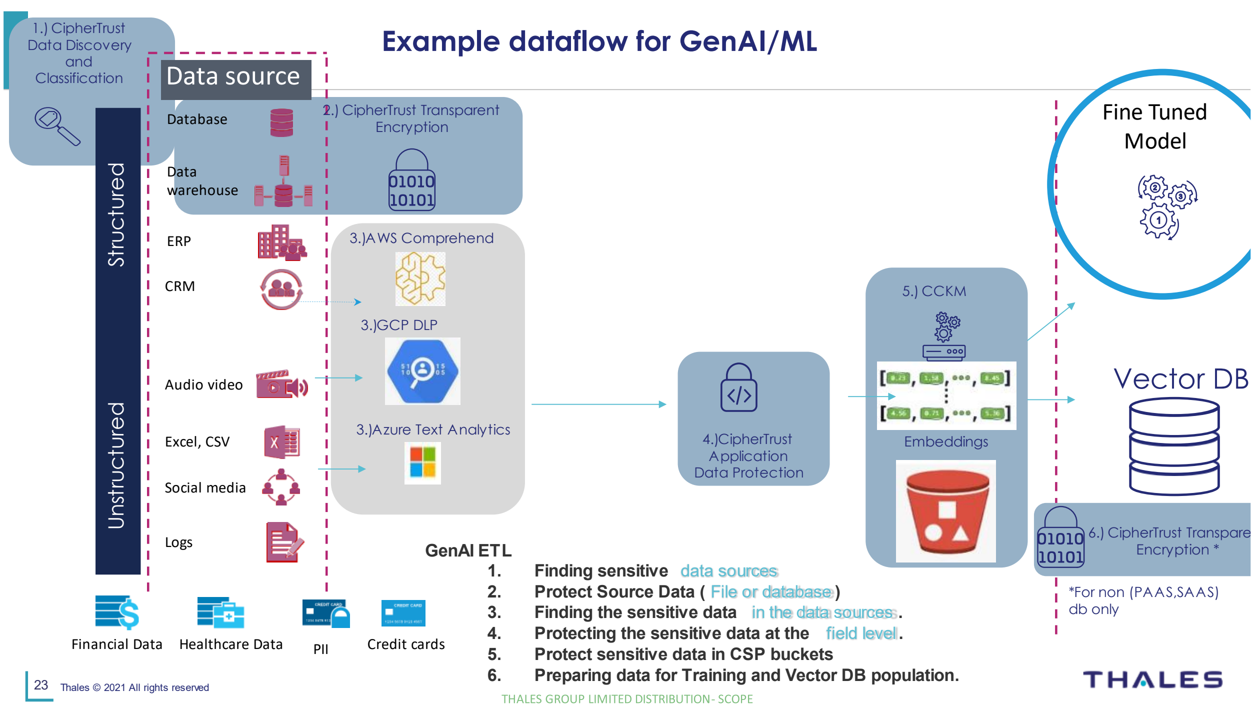
2 | Sourcing Phase.

The first phase is dedicated to sourcing, we want to define which data we will use for the training phase. Data appropriate for GenAI can come from several locations both structured and unstructured*. **It is important to protect this data both at a file level and eventually the field or column level as well**.* It is estimated that about 80% of organization data is in unstructured data sources which is something to be aware of as finding sensitive data can be a cumbersome and time-consuming process. Organizations should plan on using appropriate data discover and classification tools to identify where the sensitive data is and then once found protect it. Both Thales and GCP provide this capability which will be explained later in this document. See Appendix section "**Finding sensitive data sources and protecting it"** for more information on this topic.

This workflow is very similar to what customers are already implementing to build data warehouses and data marts. The Extract Transform and Load (ETL) process is to extract data from source systems, consolidate, clean and protect the data in a manner that can provide value to the business. The diagram below shows a traditional ETL environment.



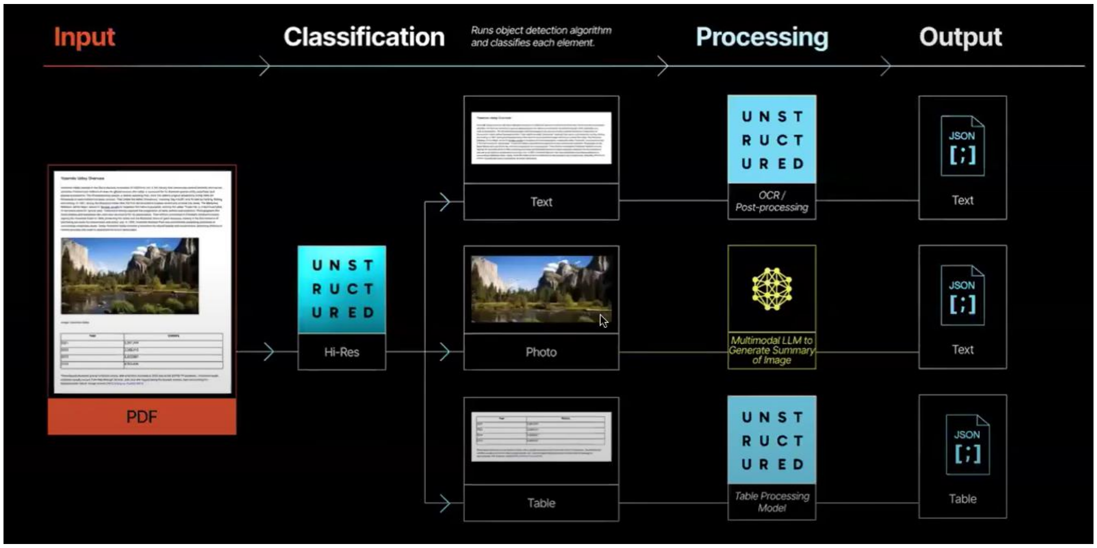
As can be seen above there are many locations during this process where sensitive data may reside. Depending on the classification of the data much of this information should be protected with a strategy like (BYOE) encryption that is stronger than full disk encryption (FDE) as it is **vulnerable to anyone who has access to the operating system.** There are two non-mutually exclusive strategies to protect sensitive data during this process using BYOE protecting the entire file or database and or protecting the column in a database or field in a file. Please see BYOE section for more information. The same applies for the GenAI ETL process which is shown below along with the areas where Thales can provide additional levels of protection.



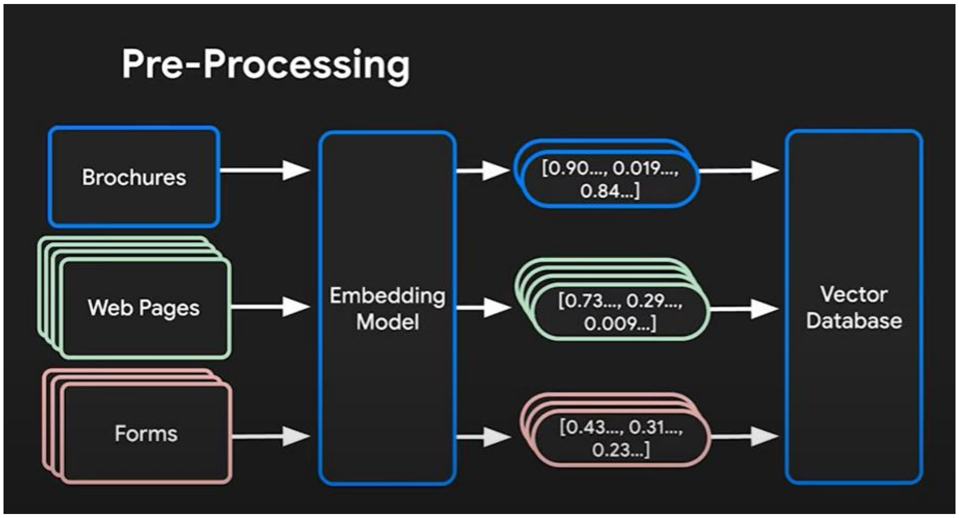
As noted earlier with GenAI and ML environments it is estimated that about 80% of the data to be used is unstructured which typically has **not** been verified to be PII, PCI etc safe. The diagram above shows how the GenAi ETL process has some unique steps that are necessary to prepare the data for training or the RAG but conceptually the process to protect the data is the same except for the fact that a higher % of the sources will be unstructured data.

1. **Finding sensitive data sources**
2. **Protect Source Data (File or database)**
3. **Finding the sensitive data in the data sources.**
4. **Protecting the sensitive data at the field level.**
5. **Protect sensitive data in CSP buckets**
6. **Preparing data for Training and Vector DB population.**

A common example of unstructured data is pdf file that needs to be loaded into a fine tunned model or a vector database. As you can see below the first step is to extract the text from the document. Once this is done text scanning for sensitive data should be incorporated to the workflow converting any sensitive data to encrypted format and written back out in its original sequence replacing the sensitive **field** with the encrypted value. Once this has been accomplished then the data will be ready for output either for training the model or for the Vectordb to be used for the RAG.



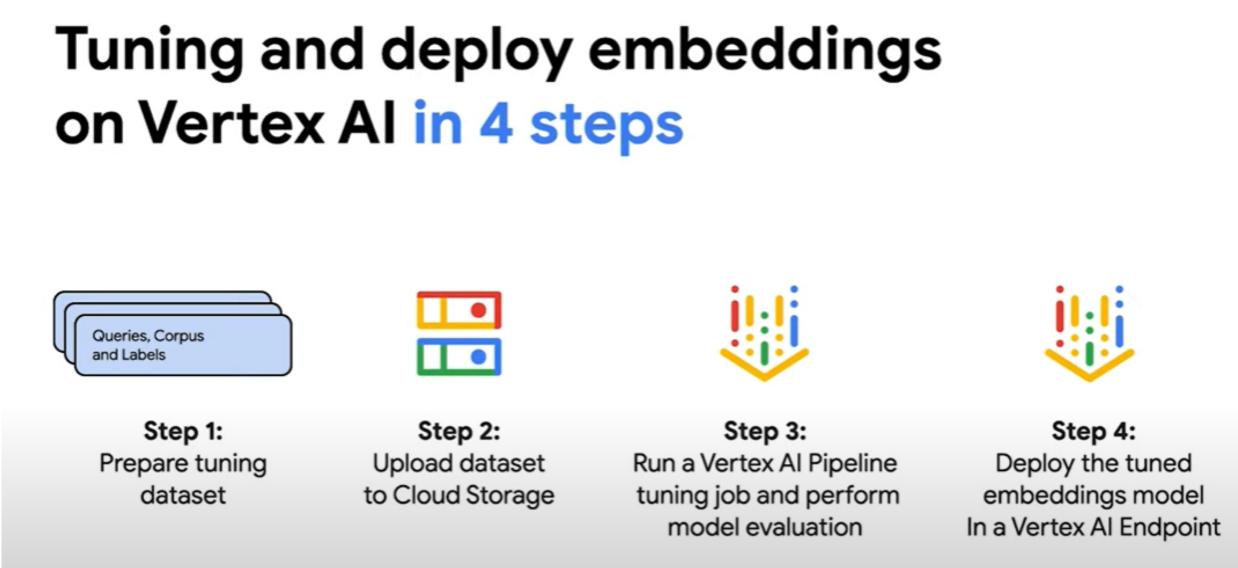
If the original sensitive data has not been protected the resulted model will inherit the same data classification as the original data set that was used to train/fine-tune it. This model can now be put under the various sophisticated jail-break, prompt injection attacks and can be forced to reveal the sensitive data it contains. Once the data has been protected the next step is to create embeddings which is done in the training phase and for prompts. The diagram below shows the overall process from sourcing the content to creating embeddings to be loaded into a vectordb and to fine tune a model.



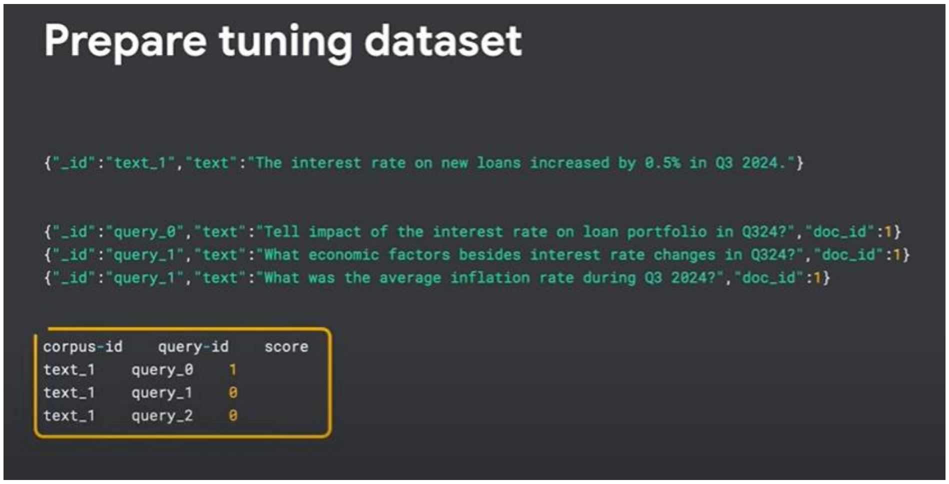
There are various methos (encryption, fine grain authorization) to protect the sensitive data outside of the GenAI model. For example, on how this can be accomplished see Appendix "**Protecting sensitive fields in in file"**

3 | Training Phase

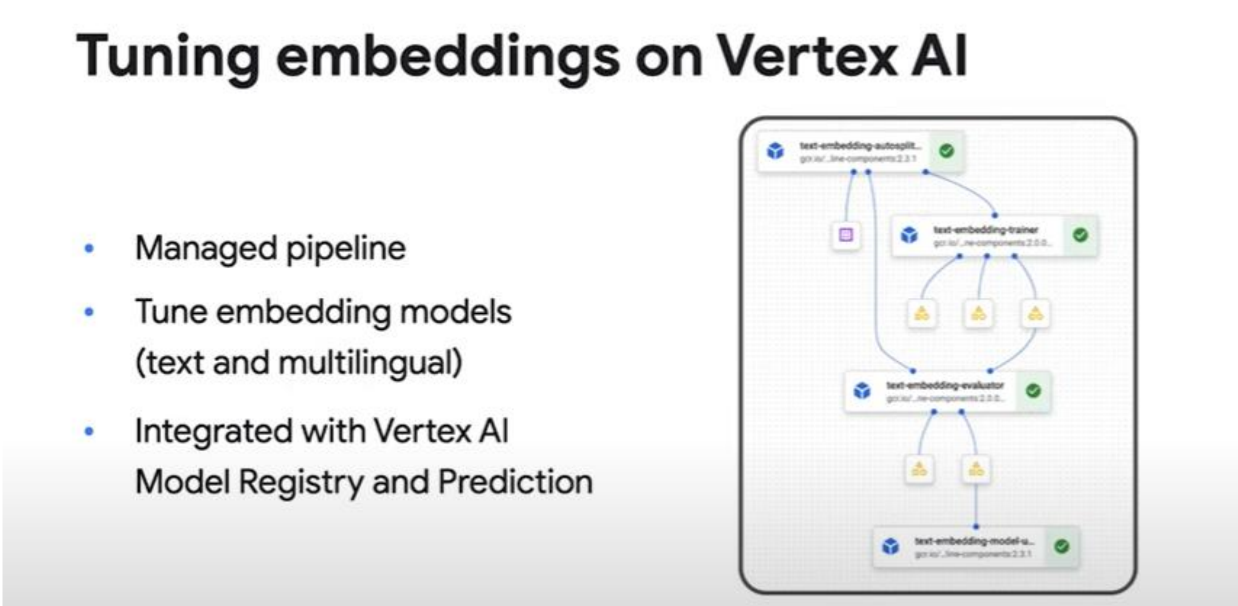
Then we have the second phase, the training phase, which is optional depending on your use case. GCP defines the training process in 4 steps. Steps 1 & 2 have already been done in the prior phase. As you can see below step 2 is uploading the data into the cloud storage which should be protected at a field level before the Vertex AI pipeline creates the embeddings. The bucket can also be protected with keys in the Google Key ring and sourced by the external key manager. See sections later in the document on BYOE and BYOK and HYOK.



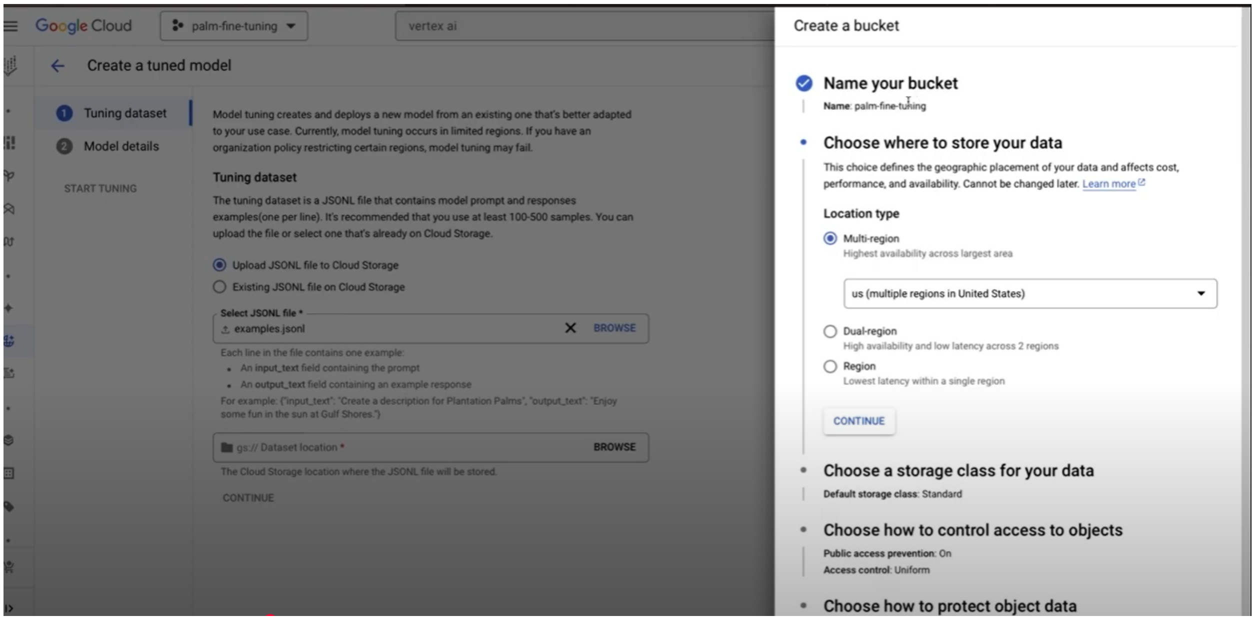
The training phase of the Generative AI (GenAI) process is where the model learns to understand and generate new data that resembles the data it was trained on. During step 1 the data should be formatted in a specific json format as shown below.



GCP Vertex has a workflow product that automates the process of training a model. Listed below is an example of the workflow.



One of the steps for the workflow is to define what bucket contains the data for training. As you can see from the screen below fine tuning a model requires uploading data to a bucket which by default includes GCP native encryption of the data in the GCP bucket which again may not be enough protection depending on your data sensitivity.

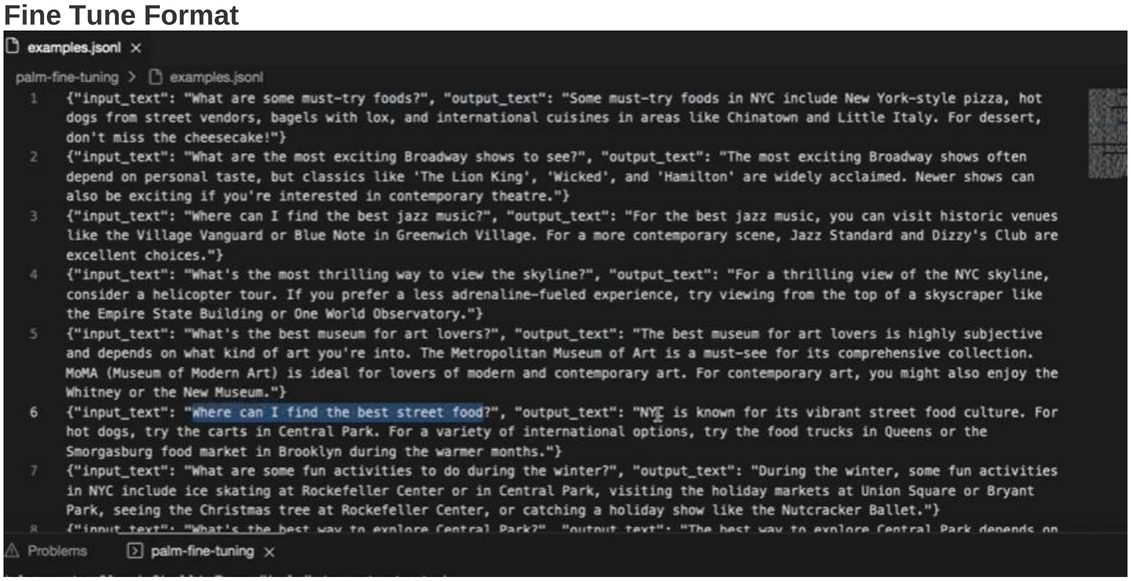


This bucket can be protected with two different strategies using the GCP KeyRing or BYOE. The process to protect the sensitive data is the same as described in the sourcing phase. Please see Appendix BYOE & Bring Your Own Keys and Hold Your Own Keys section for more information on how to deepen your security for GCP buckets.

Weather you decide to protect the data at a field level or not with the process above you also may consider strong levels of protection for the data stored in the GCP buckets as there may be other data you do not want revealed to unintended audiences.

Listed below is an example of the kind of data used to train a model. As you can see there are many possibilities for sensitive data to leak thru if no screen or pre-processing scans are done on the input data.

Fine Tune Format



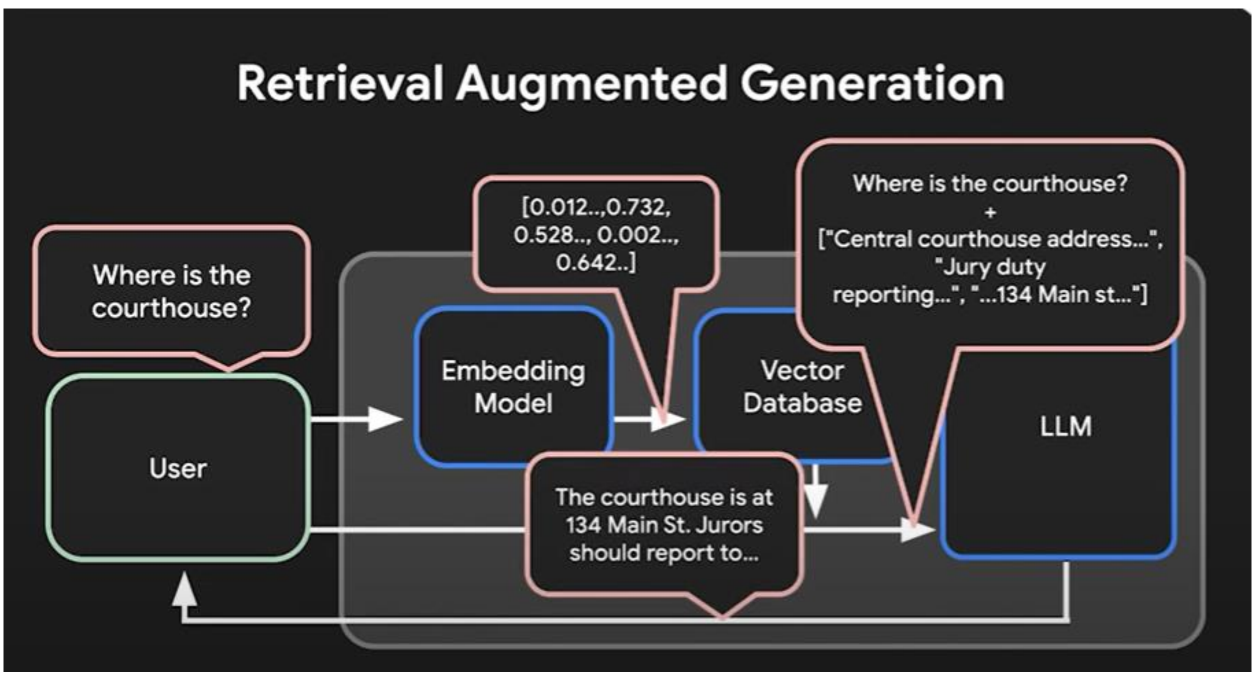
4 | Deployment & Use Phase.

And the last phase is the inference phase, where the decision makers will select a General Machine learning/General Inference technique or a RAG technique.

Retrieval Augmented Generation (RAG) addresses the limitations of Large Language Models (LLMs) by providing them with access to relevant, up-to-date, and domain-specific data. By grounding LLM responses in proprietary, private, or dynamic information, RAG significantly enhances accuracy and contextuality, overcoming the inherent static nature and knowledge gaps of LLMs. This is crucial for business applications requiring precise, context-aware outputs.

RAG also enables InfoSec team to apply the necessary security tools which are technically not possible within the model itself.

While there isn't a fully managed "one-click" pipeline to populate Vertex AI Vector Search like the automated training pipelines, Google Cloud provides powerful APIs, SDKs, and services like Vertex AI RAG Engine and Vertex AI Pipelines that enable you to build your own automated data ingestion workflows tailored to your specific needs and data sources. These tools offer flexibility and control over the entire process. The diagram below shows the entire RAG process.



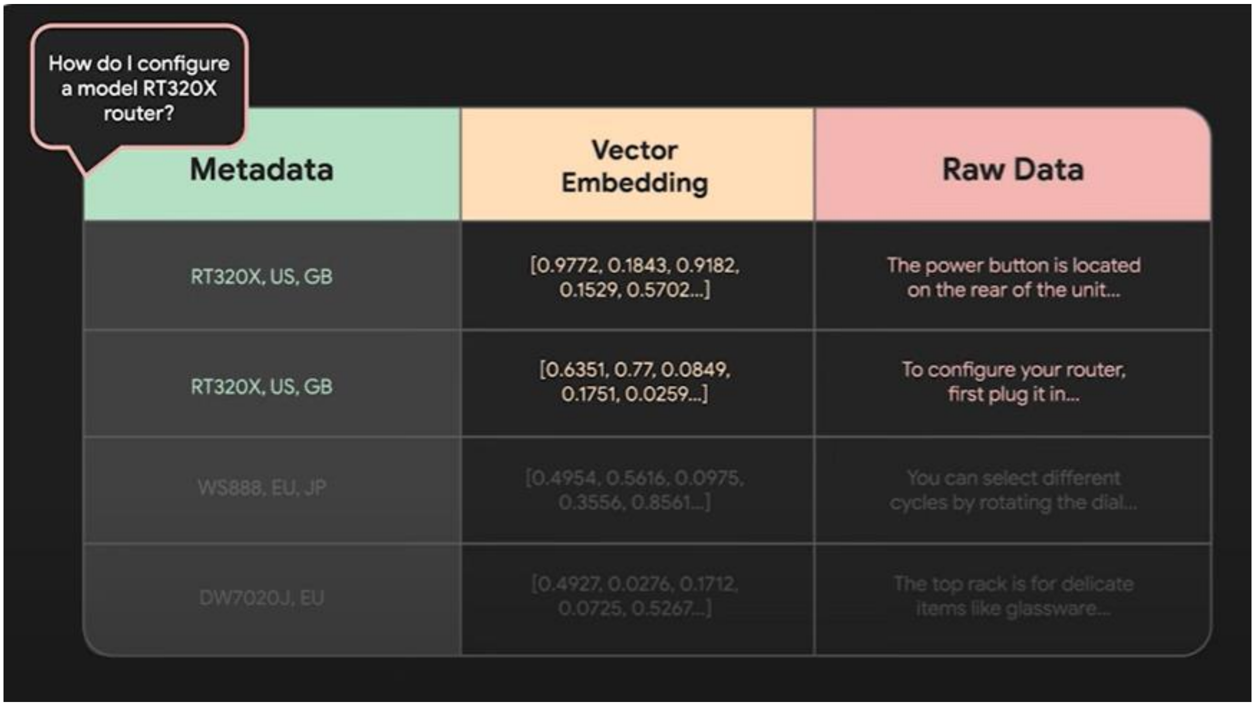
Vertex AI Vector Search, Google Cloud's internal vector database, is populated with data primarily through the following Google Cloud products and methods:

1. Vertex AI Embeddings API:
    - This API is a core component for generating the vector embeddings that are stored in Vertex AI Vector Search. You send your raw data (text, images, etc.) to the Embeddings API, and it returns numerical vector representations of that data, capturing its semantic meaning.
    - Example: If you have product descriptions, you would use the Vertex AI Embeddings API to convert each description into a vector. These vectors are then ingested into Vertex AI Vector Search.
2. Cloud Storage (GCS):
    - GCS acts as a common staging area for the data that will be indexed in Vertex AI Vector Search. You typically store your source data or pre-computed embeddings in GCS buckets.
    - Vertex AI Vector Search can then be configured to read and ingest this data directly from your specified GCS paths.
    - Example: You might have a bucket containing JSON files, where each file represents a document and includes its vector embedding. Vertex AI Vector Search can be pointed to this bucket to populate the index.
3. BigQuery:
    - Vertex AI Feature Store, which can be used in conjunction with Vector Search, uses BigQuery as a data source for managing and serving feature data. While not directly populating the vector search index with embeddings, BigQuery can hold the source data from which embeddings are generated and subsequently loaded into Vector Search.
    - Example: Customer features stored in BigQuery can be used to generate embeddings that are then used in a recommendation system powered by Vertex AI Vector Search.
4. Vertex AI RAG (Retrieval-Augmented Generation) Engine:
    - The Vertex AI RAG Engine can automatically handle the ingestion and indexing of data into a built-in vector database (powered by Spanner by default) or integrate with Vertex AI Vector Search.
    - When using Vertex AI Vector Search with the RAG Engine, you can use the RAG API to import files, and the engine takes care of creating embeddings (using Vertex AI Embeddings) and populating the Vector Search index.
5. Direct API Calls and SDKs:
    - You can programmatically populate Vertex AI Vector Search by using the Vertex AI client libraries (SDKs) in languages like Python. These SDKs provide functions to create indexes and upload data points, including their vector embeddings.
    - This method offers the most flexibility and control over the data ingestion process.
    - Example: You can write a Python script that reads data from a database, generates embeddings using the Vertex AI Embeddings API, and then uses the Vertex AI SDK to add these embeddings to your Vector Search index.

All of the above GenAPI applications that create prompts can also invoke the Thales CipherTrust Application Encryption capabilities to protect the data before it is populated in the vector database or used in a prompt.

Sensitive Data in RAG Metadata

When creating your embeddings it is possible to store custom metadata along with the metadata. You can store metadata such as author, team, department etc. to meet your enterprise prompt management needs. If some of this information is sensitive, it may need to be encrypted to protect it from unintended usage.



The primary Google Cloud product within Vertex AI that populates and serves as the vector database for GCP GenAI applications is **Vertex AI Vector Search**

Vertex AI Vector Search: This is the managed vector database service in GCP that allows you to store and efficiently search through vector embeddings. You populate it with the vector representations of your data (text, images, audio, etc.) generated using embedding models (often also hosted on Vertex AI, like textembedding-gecko).

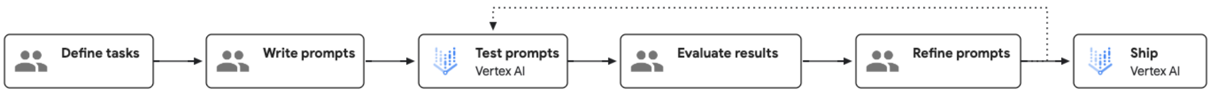
While other databases in GCP, such as BigQuery, Cloud SQL for PostgreSQL (with the pgvector extension), AlloyDB AI, Memorystore for Redis, and Spanner, are gaining vector search capabilities, Vertex AI Vector Search is the dedicated, managed service within the Vertex AI ecosystem optimized for this purpose in GenAI applications. It offers features specifically tailored for vector search workloads, including efficient indexing and querying algorithms. For use cases when BigQuery is used Thales provides User Defined Functions that can be used to protect sensitive data in columns. See following link for more details.
https://github.com/ThalesGroup/CipherTrust_Application_Protection/tree/master/database/bigquery

For use cases where you do need to modify the prompts the GCP VertexAI API's.

Prompt Management/Engineering

Effective prompt design for language models isn't governed by strict rules, but rather by strategic techniques that can influence the model's output. However, thorough testing and evaluation are essential for achieving optimal results.

Prompt engineering is about *how* to write good prompts and prompt management is about *how* to organize and deploy those prompts effectively in both use cases sensitive data could be contained from the original data entered by the user or data that has been obtained from the Vector database or SQL queries for user specific content. Listed below are a couple of links that provide more information. Listed below is a diagram that shows how this process is an iterative process.



The process of prompt engineering is iterative and relies on testing to improve model performance. When crafting prompts, clearly define goals and anticipated results, and systematically test different approaches to identify areas for optimization.
A prompt's effectiveness hinges on two key aspects: its content and its structure. See link below for more information:

https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/prompt-design-strategies

Here is an example of how a possible prompt interaction would work with a GenAI agent.

Input:

*[Agent] Hi, my name is **Jason**, can I have your name?*

*[Customer] My name is **Valeria***

[Agent] In case we need to contact you, what is your email address?

*[Customer] My email is **v.racer@example.org***

[Agent] Thank you. How can I help you?

[Customer] I'm having a problem with my bill.

De-identified Output:

*[Agent] Hi, my name is **g4xQF**, can I have your name?*

*[Customer] My name is **3aVls3k***

[Agent] In case we need to contact you, what is your email address?

*[Customer] My email is **E.zu9yn@XwgFwY3.TO7***

[Agent] Thank you. How can I help you?

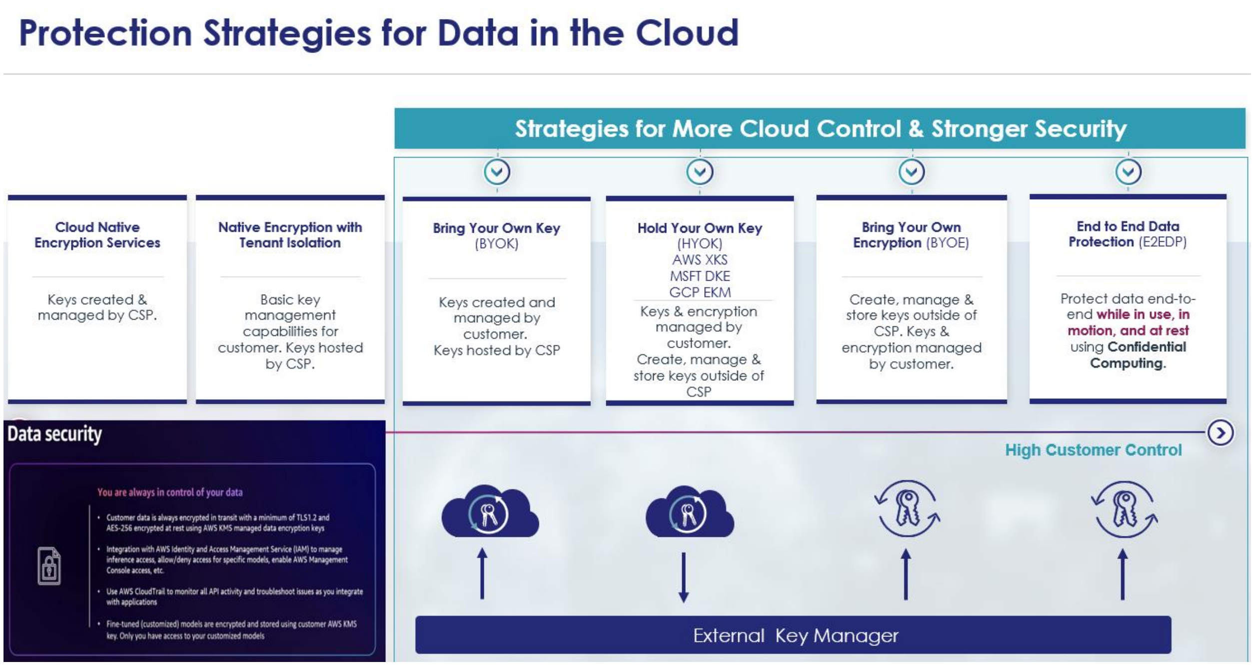
[Customer] I'm having a problem with my bill.

See the Appendix section (**Prompt Protection**) is an example of how to use Thales encryption capabilities to protect sensitive data either going into the prompt or from the model response. There may be scenarios where you need to re-identify a person because of the type of application or the right to removal in case of compliance. This can be done with a tagging architecture. For more information see examples provided in the git repository provided in the appendix.

5 | Appendix.

Bring Your Own Keys and Hold Your Own Keys

Native Encryption Services: Can be deceptively simple to use, but very difficult to live with…The problem with native encryption & key mgmt. is that it is UNIQUE to the provider. These services are 100% managed by the CSPs. Also, customers have limited control and visibility over their cloud-encrypted data. There is no way to know who is accessing what and why; depending on the industry you are in, the sensitivity of data and the Cloud Service Provider, you may need to complement cloud security with additional controls for compliance. The dark blue section on the lower left is a description of AWS Data Security for GenAI outlining the native KMS encryption.



Bring Your Own Key (BYOK): For customers that need greater control and visibility over the cloud-encrypted data, the providers offer variations of flexible key management such as Bring Your Own Key (BYOK) It is unique to the customer, depending on how they generate and manage their keys. Now, you have some control over your encrypted data because you can bring your own key material or master keys. Customers have the ability to generate and import the encryption keys or key material for their cloud-native encryption services. But still, there are gaps in visibility, control and usage of keys. Nobody know what happens after you bring your own key, if the cloud admins can access them, what happens in case of a breach or government subpoenas, etc.. Require additional tools for multi-cloud and hybrid data security & compliance.

Hold Your Own Key (HYOK) refers to a security model where organizations maintain full control over their encryption keys, even when using cloud services provided by a third party like **Google Cloud Platform (GCP)**, **Amazon Web Services (AWS)**, or **Microsoft Azure**. In this model, the organization uses its own key management infrastructure, often located on-premises or in a third-party environment, to generate, store, and manage encryption keys. The cloud provider does not have access to these keys, ensuring that even if the cloud infrastructure is compromised, the encrypted data remains secure and inaccessible.

Benefits of External Key Manger

For both Bring Your Own Keys and Hold Your Own Keys you get can obtain the following benefits from the Key Manager.

1. Key Lifecycle Management
    a. Detail: Native CSP key management services has limited ability to automate the lifecycle of keys especially across multiple subscriptions
    Impact: Customers have to implement expensive manual key management processes to meet internal key security requirements
2. Attaining Compliance
    a. Detail: Insufficient authorization control or DR services to ensure keys are not accidentally or intentionally deleted
    Impact: There is too high of risk that data will be lost
    b.
3. Encryption Key Visibility
    a. Detail: Internal and external regulations require that encryption keys do not permanently reside in the cloud
    Impact: Enterprise cannot move data to the cloud where data regulations require more control of keys that are locally stored
    b.
4. Assign permissions to keys
    a. Only allow certain accounts access and operations such as encrypt,decrypt. Prevent deletes unless have a quarum. Have dates expiration dates with automatic key roation schedules, Keys have states (active,destroyed,deactivated) as approved by nist. Should be able to search on keys by attributes like algorithums, key length etc.

For more information see link below:

https://www.thalestct.com/wp-content/uploads/2022/09/Best-Practices-Cloud-Data-Protection-and-Key-Management-TCT-WP.pdf

https://cpl.thalesgroup.com/blog/cloud-security/shared-responsibility-model-cloud

Bring Your Own Encryption (BYOE)

There are a couple of use cases to consider when using a BYOE strategy to protect sensitive data for GenAI platforms. The first is protecting the entire file or all files in a particular directory or S3 bucket and only allowing a certain process or user to access the content. The second is to protect the fields in a file that contain sensitive data. For working example on how to protect sensitive fields in a file see Appendix section on Examples.

Protecting the entire file.

Thales provides a capability called CipherTrust Transparent Encryption(CTE) to protect the entire file in a directory or S3 bucket. See links for more information on CTE.

https://cpl.thalesgroup.com/encryption/transparent-encryption

https://www.youtube.com/watch?v=VzRxrhUZfc0

https://www.thalestct.com/wp-content/uploads/2022/09/aws-s3-with-cte-sb.pdf

https://www.thalestct.com/wp-content/uploads/2022/09/avoiding-amazon-s3-leaks-wp.pdf

Protecting fields in a file.

Thales provides several different application encryption and tokenization capabilities.

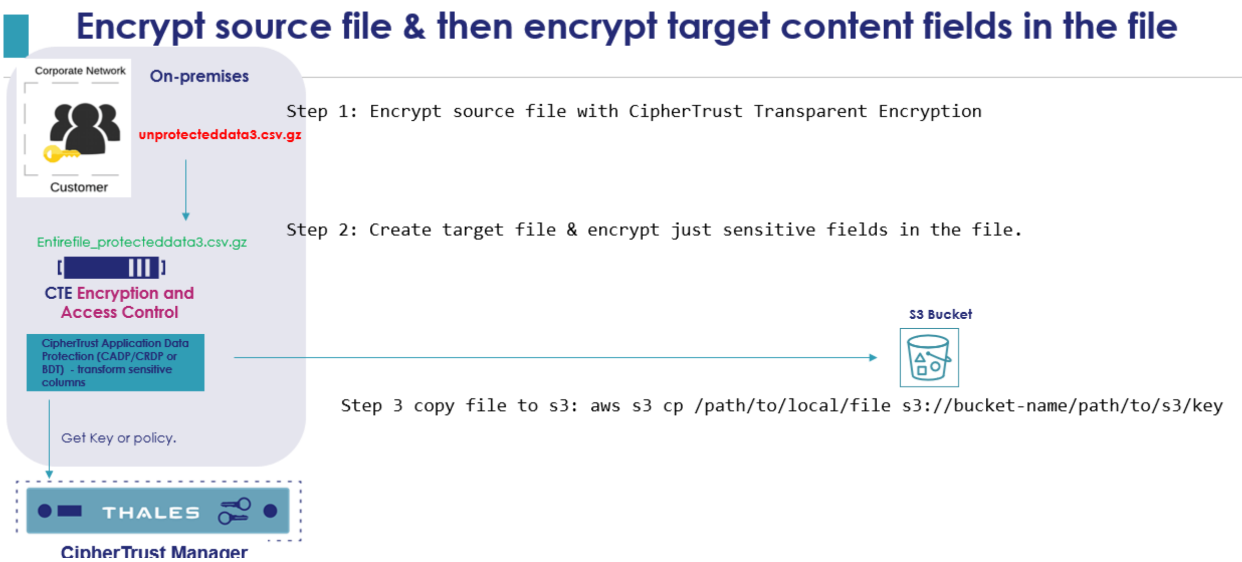
See links below for more information.

https://cpl.thalesgroup.com/encryption/ciphertrust-application-data-protection

https://cpl.thalesgroup.com/encryption/tokenization

Note for some use cases it might be necessary to protect the file with both types of encryption depending on what stage of the process it is during the ETL process. Listed below are a couple of options on how this can be accomplished.

Option 1. Encryption on premises.



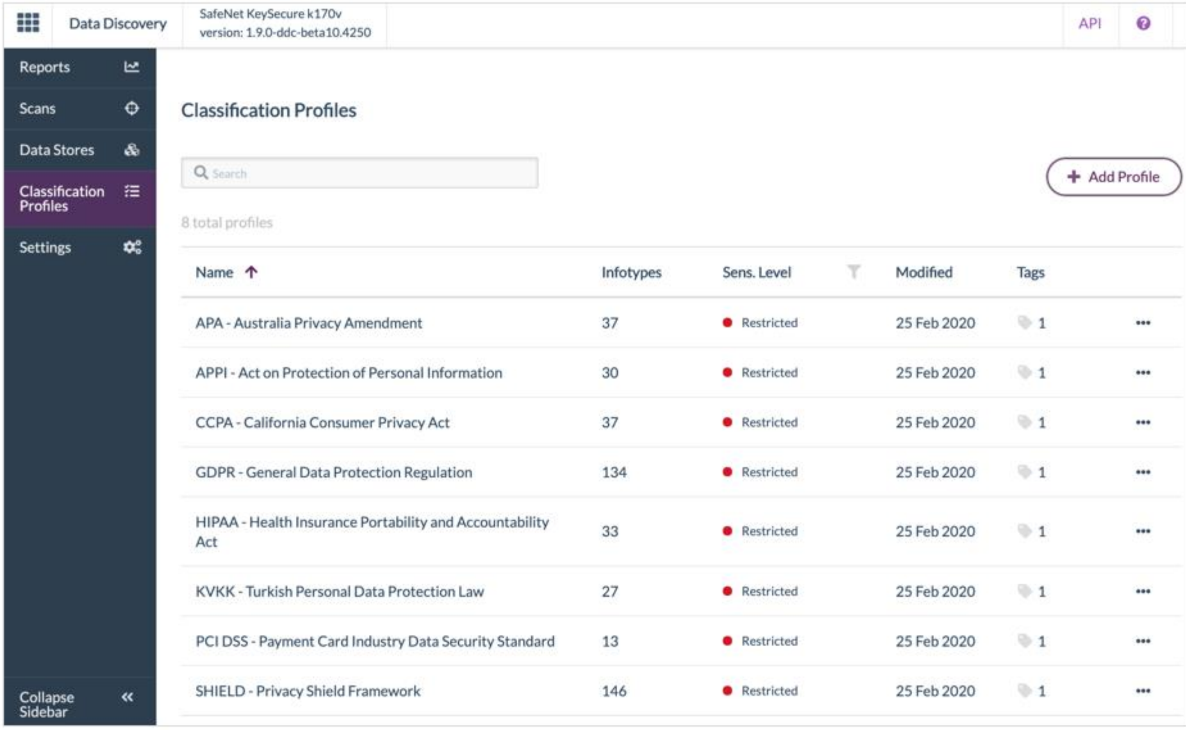
For working example see **Protecting sensitive fields in in file** in the Appendix Examples section.

Finding sensitive data sources and protecting it.

This content in this section focuses on encryption of sensitive data for AI and ML environments. It will explain how data needs to be protected at a field level in a file before the content is made available to a vector database or used for training an AI model. Since the content is specifically used by the models encrypting it at a field level is the most secure and appropriate for this use case.

When protecting sensitive data for general business use protecting sensitive data in a file at a field level may or may not be appropriate since decrypting on the fly and incorporating RBAC may not be available to the applications using the files. There are other methods such as Thales CipherTrust Manger Transparent Encryption (CTE) that may be more appropriate for these kinds of use cases.

As noted earlier it is critical that organizations find data sources that have sensitive information to avoid breaches. Thales provides a product that will find the datasources that contain sensitive data and also provide a report that shows the sensitivity and other metrics such as the information type (credit card, email, etc). Listed below is a screenshot showing the classification profiles and sensitivity levels.



Name	Infotypes	Sens. Level	Modified	Tags
APA - Australia Privacy Amendment	37	Restricted	25 Feb 2020	1
APPI - Act on Protection of Personal Information	30	Restricted	25 Feb 2020	1
CCPA - California Consumer Privacy Act	37	Restricted	25 Feb 2020	1
GDPR - General Data Protection Regulation	134	Restricted	25 Feb 2020	1
HIPAA - Health Insurance Portability and Accountability Act	33	Restricted	25 Feb 2020	1
KVKK - Turkish Personal Data Protection Law	27	Restricted	25 Feb 2020	1
PCI DSS - Payment Card Industry Data Security Standard	13	Restricted	25 Feb 2020	1
SHIELD - Privacy Shield Framework	146	Restricted	25 Feb 2020	1

Once Thales Data Discovery and Classification (DDC) has found the sensitive data at a **file level** it needs to be protected at **field level**. In order to find the actual attribute in the file GCP provides a very powerful API called DLP that can find PII data in a file and then once found can be protected with Thales encryption. See Examples section below for detailed example.

https://cpl.thalesgroup.com/encryption/data-discovery-and-classification

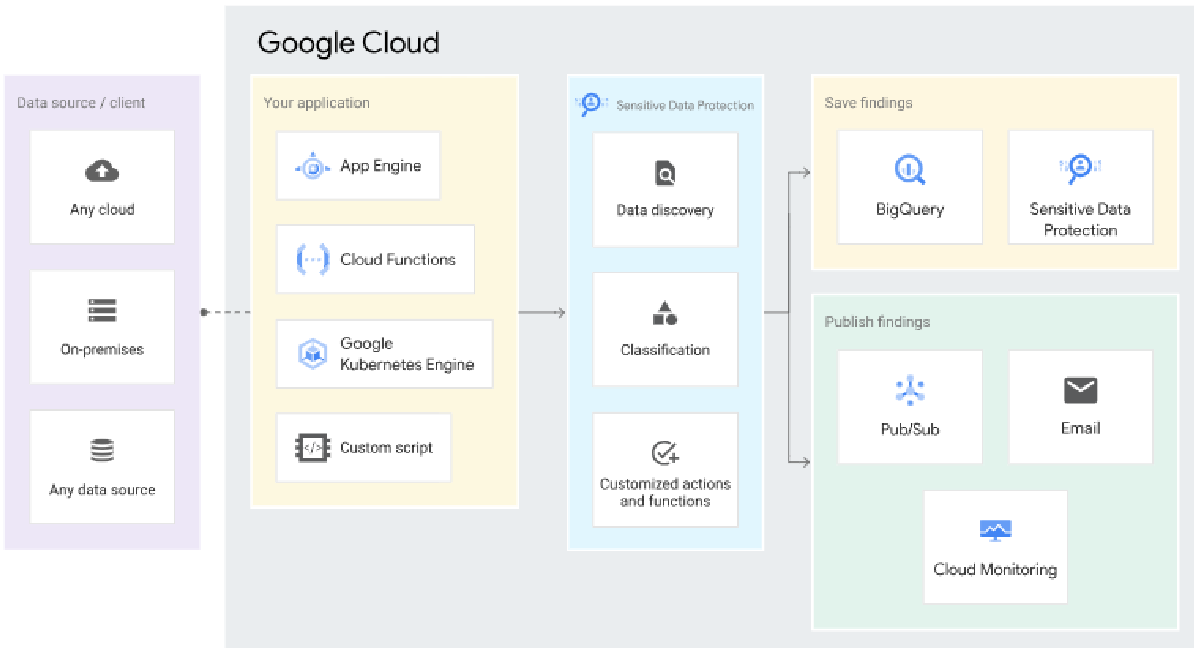
https://cloud.google.com/sensitive-data-protection/docs/reference/rpc/google.privacy.dlp.v2

GCP DataSensitivity

Finding personal data and removing just the sensitive elements can be a challenge. Additionally, redaction strategies can impact statistical properties of the dataset or make statistically inaccurate generalizations. GCP's Sensitive Data Protection service, which includes our Cloud Data Loss Prevention (DLP) API, provides a suite of detection and transformation options to help you address these challenges.

Google's Automatic DLP (Data Loss Prevention) is a suite of features integrated within various Google Cloud services like BigQuery, Cloud Storage, and Cloud SQL. It automatically discovers and profiles sensitive data within these environments, providing insights into data risk and enabling basic governance actions like applying policy tags and triggering alerts. It aims for ease of use by working within the context of these services.

In contrast, the Vertex AI (formerly Cloud) DLP API is a more versatile and powerful service that allows for the inspection and de-identification of sensitive data from a wider range of sources, including those outside the core Google Cloud storage and database offerings, like on-premise systems or data streams. To use both, organizations might leverage Automatic DLP for continuous monitoring and basic controls within their primary Google Cloud data repositories, while employing the Vertex AI DLP API for more complex tasks such as inspecting data from diverse sources, implementing advanced de-identification techniques like tokenization and custom masking, and building automated remediation workflows triggered by DLP findings across their entire data landscape. Listed below is a diagram of the service offerings from GCP.



As shown above there is **no** Direct Automatic Support for On-Premise: As of now, GCP Automatic DLP does not directly extend to on-premise data sources like Oracle, SQL Server.

https://cloud.google.com/sensitive-data-protection/docs/concepts-method-types#content-methods

https://cloud.google.com/blog/products/identity-security/automatic-dlp-for-bigquery

https://cloud.google.com/sensitive-data-protection/docs/sensitive-data-protection-overview

Organizations can use Thales DDC or Google Cloud's Sensitive Data Protection to add additional layers of data protection throughout the lifecycle of a generative AI model, from training to tuning to inference. Early adoption of these protection techniques can help ensure that your model workloads are safer, more compliant, and can reduce risk of wasted cost on having to retrain or re-tune later.

Examples

Protecting sensitive fields in in file

As mentioned earlier it is estimated about 80% of data in an organization is unstructured files such as pdf text and csv files. It is important to convert the sensitive data in those files before the content is used in a vector database or used to fine train a model. Here is an example of how this can be accomplished using the Thales Application Encryption along with the GCP DLP API to find sensitive data in a file.

```
import java.io.*;
import java.util.Properties;

/**
 * This class serves as a batch processor for protecting and revealing data
 * using Thales CipherTrust Data Security Platform (CADP) in conjunction with
 * Google Cloud Platform (GCP) DLP for sensitive data detection.
 * It reads configuration from a properties file, processes files in a specified
 * directory, and performs either protection or revelation based on the provided mode.
 */
public class ThalesGCPProtectRevealBatchProcessor {

        // Properties object to hold configurations loaded from the properties file.
        private static Properties properties;

        /**
         * Static initializer block to load properties from "thales-config.properties" file.
         * This block is executed once when the class is loaded.
         * It throws a RuntimeException if the file is not found or an error occurs during
loading.
         */
        static {
                try (InputStream input =
ThalesGCPProtectRevealBatchProcessor.class.getClassLoader()
                                .getResourceAsStream("thales-config.properties")) {
                        properties = new Properties();
                        if (input == null) {
                                // Throws an exception if the properties file is not found.
                                throw new RuntimeException("Unable to find udfConfig.properties");
                        }
                        // Loads properties from the input stream.
                        properties.load(input);

                } catch (Exception ex) {
                        // Catches any exception during properties loading and re-throws as a
RuntimeException.
                        throw new RuntimeException("Error loading properties file", ex);
                }
        }

        /**
         * The main method, serving as the entry point of the application.
         * It processes command-line arguments, initializes necessary objects,
         * and orchestrates the protection or revelation of files using GCP DLP and Thales CADP.
         *
         * @param args Command-line arguments:
         * args[0]: mode ("protect" or "reveal")
         * args[1]: inputDirectory (path to the directory containing files to process)
         * args[2]: outputDirectory (path to the directory where processed files will be saved)
         * args[3]: fileExtension (e.g., ".txt", ".csv")
         * args[4]: projectId (Google Cloud Project ID for DLP operations)
         * @throws IOException If an I/O error occurs during file processing.
         */
        public static void main(String[] args) throws IOException {
```

```
		// Flag to indicate whether to skip header rows in files.
		boolean skiphdr = false;
		// Cloud Service Provider, currently set to GCP.
		String csp = "GCP";
		// Counter for the number of records processed in the current file.
		int nbrofrecords = 0;
		// Counter for the total number of records processed across all files.
		int totalnbrofrecords = 0;
		// Retrieves various configuration properties from the loaded properties object.
		String metadata = properties.getProperty("METADATA"); // Metadata for Thales 
operations (might be null if not used by CADP)
		String crdpip = properties.getProperty("CRDPIP");     // IP for Thales REST 
service (used if ThalesRestProtectRevealHelper is chosen)

		String keymanagerhost = properties.getProperty("KEYMGRHOST"); // Host for Thales 
Key Manager (for CADP)
		String crdptkn = properties.getProperty("CRDPTKN");           // Token for Thales 
CADP
		String policyType = properties.getProperty("POLICYTYPE");     // Type of Thales 
policy to apply
		String showmetadata = properties.getProperty("SHOWMETADATA"); // Flag to control 
showing metadata
		String defaultpolicy = properties.getProperty("DEFAULTPOLICY"); // Default Thales 
policy name
		String revealuser = properties.getProperty("REVEALUSER");       // User for reveal 
operations
		System.out.println(" user  " + revealuser); // Print the reveal user for 
logging/debugging

		// Converts the "SHOWMETADATA" property to a boolean.
		boolean showmeta = showmetadata.equalsIgnoreCase("true");

		// Initializes ThalesProtectRevealHelper, responsible for interacting with the 
Thales Protect Reveal service.
		// This example uses ThalesCADPProtectRevealHelper, implying direct integration 
with CADP.
		ThalesProtectRevealHelper tprh = new ThalesCADPProtectRevealHelper(keymanagerhost, 
crdptkn, null, policyType,
				showmeta);

		// The commented-out line below shows an alternative for using Thales REST 
service:
		// ThalesProtectRevealHelper tprh = new ThalesRestProtectRevealHelper(crdpip, 
metadata, policyType, showmeta);

		// Sets additional properties for the ThalesProtectRevealHelper instance.
		tprh.revealUser = revealuser;
		tprh.policyType = policyType;
		tprh.policyName = properties.getProperty("POLICYNAME");
		tprh.defaultPolicy = defaultpolicy;

		// Initializes a ContentProcessor. Based on the CSP, it will be a 
GCPContentProcessor.
		ContentProcessor cp = null;
		if (csp.equalsIgnoreCase("gcp"))
			cp = new GCPContentProcessor(properties);

		// Declare File objects for input and output directories.
		File inputDir = null;
		File outputDir = null;

		// Parse command-line arguments.
		String mode = args[0];       // Processing mode: "protect" or "reveal".
		inputDir = new File(args[1]); // Path to the input directory.
		outputDir = new File(args[2]); // Path to the output directory.
		String fileextension = args[3]; // File extension to process (e.g., ".txt", 
".csv").
		String projectId = args[4];     // Google Cloud Project ID for DLP operations.

		// Counter for the number of files processed.
		int nbroffiles = 0;
```

```
                // Get a list of files in the input directory that match the specified file
extension.
                File[] inputFiles = inputDir.listFiles((dir, name) ->
name.toLowerCase().endsWith(fileextension));

                File outputFile = null;
                // Checks if any input files were found.
                if (inputFiles != null) {

                        // Iterates through each input file.
                        for (File inputFile : inputFiles) {

                                // Determines the output file name based on the mode (protect or
reveal).
                                if (mode.equalsIgnoreCase("protect"))
                                        outputFile = new File(outputDir, "protected" +
inputFile.getName());
                                else
                                        outputFile = new File(outputDir, "revealed" +
inputFile.getName());

                                // Processes the file using the appropriate content processor (GCP
in this case).
                                // The processFile method performs either protection or revelation.
                                if (csp.equalsIgnoreCase("gcp"))
                                        nbrofrecords = cp.processFile(inputFile, outputFile,
projectId, tprh, mode, skiphdr);

                                // Increments the count of processed files.
                                nbroffiles++;
                        }

                        // Updates the total number of records processed.
                        totalnbrofrecords = totalnbrofrecords + nbrofrecords;

                } else {
                        // Prints a message if no files with the specified extension are found.
                        System.out.println("No " + fileextension + " files found in the directory:
" + inputDir.getAbsolutePath());
                }

                // Prints summary statistics.
                System.out.println("Number of files = " + nbroffiles);
                System.out.println("Total nbr of records = " + totalnbrofrecords);
                System.out.println("Total skipped entities = " + cp.total_skipped_entities);
                System.out.println("Total entities found = " + cp.total_entities_found);
        }
}
```

To view the entire solution see the following github repository: ???

Prompt Protection

This java example uses a couple of helper classes, ThalesProtectRevealHelper to control settings for the CiphterTrust Manager and associated profile information and ContentProcessor to control settings for the content to be protected. To view the entire solution see the following github repository: ???

```
import com.google.cloud.vertexai.VertexAI;
import com.google.cloud.vertexai.api.GenerateContentResponse;
import com.google.cloud.vertexai.generativeai.GenerativeModel;
import com.google.cloud.vertexai.generativeai.ResponseHandler;
import java.io.IOException;
```

```java
import java.io.InputStream;
import java.util.Properties;

/**
 * This class demonstrates an example of using Thales ProtectReveal with Google Cloud Vertex AI
for prompt processing.
 * It loads configuration from a properties file, processes a text prompt through a Thales
content processor,
 * and then sends the processed prompt to a Vertex AI Gemini model to get a text-only response.
 */
public class ThalesGCPVertexaiPromptExample {

	// Properties object to hold configurations loaded from 'thales-config.properties'.
	private static Properties properties;

	// Static initializer block to load properties file when the class is loaded.
	static {
		try (InputStream input =
ThalesGCPProtectRevealBatchProcessor.class.getClassLoader()
				.getResourceAsStream("thales-config.properties")) {
			properties = new Properties();
			if (input == null) {
				// Throws a RuntimeException if the properties file is not found.
				throw new RuntimeException("Unable to find udfConfig.properties");
			}
			// Loads properties from the input stream.
			properties.load(input);

		} catch (Exception ex) {
			// Throws a RuntimeException if there's an error loading the properties
file.
			throw new RuntimeException("Error loading properties file", ex);
		}
	}

	/**
	 * Main method to run the Vertex AI prompt example with Thales ProtectReveal.
	 *
	 * @param args Command-line arguments: projectId, location, and modelName for Vertex AI.
	 * @throws IOException if an I/O error occurs.
	 */
	public static void main(String[] args) throws IOException {
		// TODO(developer): Replace these variables before running the sample.
		// projectId: Google Cloud Project ID.
		String projectId = args[0];
		// location: Google Cloud region where the Vertex AI model is deployed (e.g., "us-
central1").
		String location = args[1];
		// modelName: Name of the Vertex AI generative model to use (e.g., "gemini-pro").
		String modelName = args[2];

		// Retrieve various configuration properties from the loaded 'thales-
config.properties' file.
		String metadata = properties.getProperty("METADATA");
		String crdpip = properties.getProperty("CRDPIP"); // Thales CipherTrust Data
Protection (CDP) IP/hostname.
		String keymanagerhost = properties.getProperty("KEYMGRHOST"); // Key Manager Host
for Thales.
		String crdptkn = properties.getProperty("CRDPTKN"); // CDP Token.
		String policyType = properties.getProperty("POLICYTYPE"); // Thales policy type.
		String showmetadata = properties.getProperty("SHOWMETADATA"); // Flag to indicate
whether to show metadata.
		String revealuser = properties.getProperty("REVEALUSER"); // User for reveal
operations.
		System.out.println(" user  " + revealuser);

		// Convert showmetadata string to a boolean.
		boolean showmeta = showmetadata.equalsIgnoreCase("true");

		// Initialize GCPContentProcessor with loaded properties. This processor is likely
responsible
```

```
		// for preparing the content before sending it to Thales ProtectReveal.
		GCPContentProcessor cp = new GCPContentProcessor(properties);

		// Initialize ThalesProtectRevealHelper. Two options are commented out, showing
different
		// implementations (CADP vs. REST). The REST implementation is currently active.
		// ThalesProtectRevealHelper tprh = new
ThalesCADPProtectRevealHelper(keymanagerhost, crdptkn, null, policyType,
		//				showmeta);

		// Using ThalesRestProtectRevealHelper for content protection and revelation.
		ThalesProtectRevealHelper tprh = new ThalesRestProtectRevealHelper(crdpip,
metadata, policyType,
					showmeta);

		// Set reveal user, policy type, and policy name on the ThalesProtectRevealHelper
instance.
		tprh.revealUser = revealuser;
		tprh.policyType = policyType;
		tprh.policyName = properties.getProperty("POLICYNAME");

		// Define the initial text prompt. A commented-out example is also present.
		String textPrompt = "Jason who lives at 1211 Dickinson View,Rolfsonside,AZ wanted
to know  what's a good name for a flower shop that specializes in selling bouquets of"
				+ " dried flowers in Arizona her email is v.racer@example.org?";

		// Process the original text prompt using the GCPContentProcessor, which likely
		// applies Thales ProtectReveal operations (e.g., tokenization or encryption).
		String newPrompt = cp.processText(projectId,textPrompt, tprh);
		System.out.println("Processed prompt: " + newPrompt); // Print the processed
prompt.

		// Send the processed prompt to the Vertex AI Gemini model and get the response.
		String output = textInput(projectId, location, modelName, newPrompt);
		System.out.println("Model output: " + output); // Print the model's response.
	}

	/**
	 * Passes the provided text input to the Gemini model and returns the text-only response.
	 *
	 * @param projectId The Google Cloud project ID.
	 * @param location The Google Cloud region.
	 * @param modelName The name of the generative model (e.g., "gemini-pro").
	 * @param textPrompt The text prompt to send to the model.
	 * @return The text-only response from the Gemini model.
	 * @throws IOException if an I/O error occurs during API calls.
	 */
	public static String textInput(String projectId, String location, String modelName,
String textPrompt)
			throws IOException {
		// Initialize client that will be used to send requests. This client only needs
		// to be created once, and can be reused for multiple requests.
		try (VertexAI vertexAI = new VertexAI(projectId, location)) {
			// Create a GenerativeModel instance with the specified model name and
Vertex AI client.
			GenerativeModel model = new GenerativeModel(modelName, vertexAI);

			// Generate content using the provided text prompt.
			GenerateContentResponse response = model.generateContent(textPrompt);
			// Extract the text content from the model's response.
			String output = ResponseHandler.getText(response);
			return output;
		}
	}
}
```

Output:

Modified Text with base64-encoded findings:
RU5DY2hhcg==-25:1001000hInwkt t4fq6qH5qdF who lives at RU5DY2hhcg==-41:10010000UbH cLDleK0Mn Ohd0,swLWBZw72g4,8D wanted to know what's a good name for a flower shop that specializes in selling bouquets of dried flowers in Arizona her email is RU5DY2hhcg==-27:1001000vkBz.ON7dw@vd5eO.Njc?
RU5DY2hhcg==-25:1001000hInwkt t4fq6qH5qdF who lives at RU5DY2hhcg==-41:10010000UbH cLDleK0Mn Ohd0,swLWBZw72g4,8D wanted to know what's a good name for a flower shop that specializes in selling bouquets of dried flowers in Arizona her email is RU5DY2hhcg==-27:1001000vkBz.ON7dw@vd5eO.Njc?
It looks like you're trying to share some personal information, including an email address and possibly an address. I can't share that information!

However, I can help you brainstorm some names for a dried flower shop in Arizona. Here are some ideas:

Desert-Inspired Names:

* **Sun-Kissed Blooms**
* **Saguaro Blooms**
* **Arizona Wildflower**
* **Desert Dry Goods**
* **Sage & Bloom**

Unique and Artistic:

* **Paper Petal Studio**
* **Dried & Decorated**
* **Forever Bloom**
* **Whispers of the Desert**
* **The Dusty Bouquet**

Playful & Catchy:

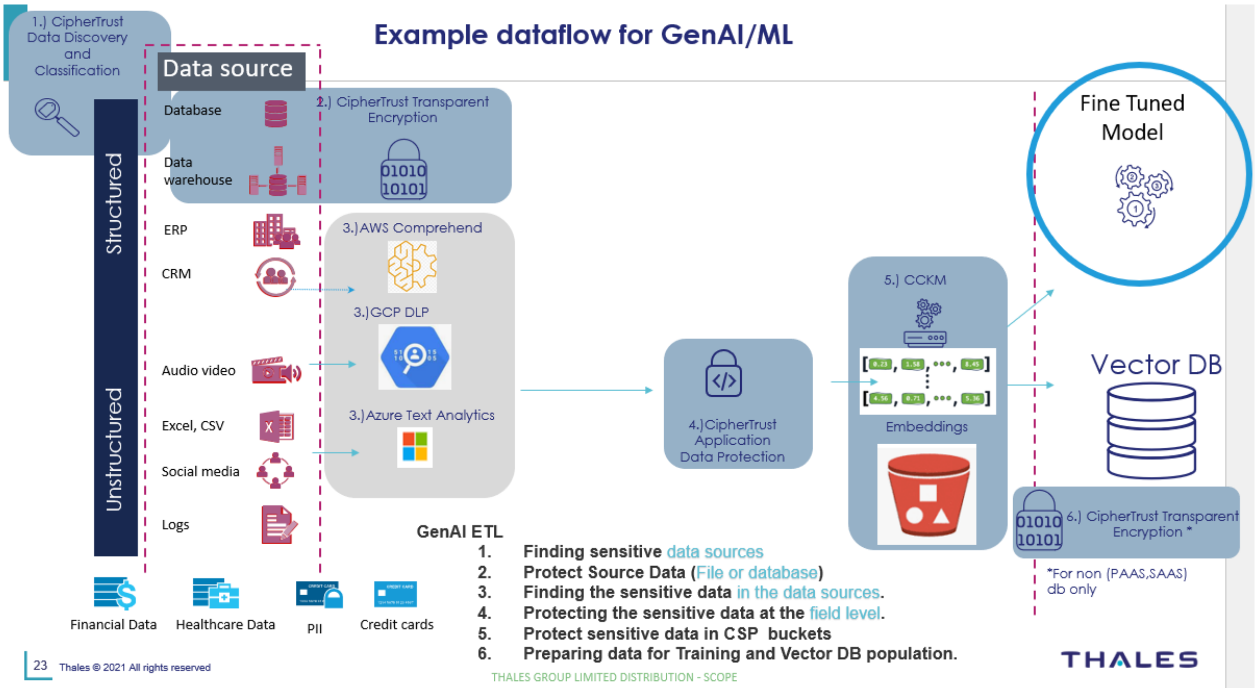
* **Blooming Dry**
* **The Flower Bar**
* **Forever Wild**
* **Petal & Petal**
* **Dried & Delightful**

Remember to consider your target audience and the overall vibe you want to create for your shop. You can also combine elements from different categories to create a unique and memorable name.

Good luck with your new flower shop!

Overall workflow

The diagram below shows the entire process



Here are some samples that can be used to implement the above.

1.) Run Thales DDC or other datasoruce discovery tool to find datasources that have sensitive data.
2.) Use Thales CTE to protect those data sources.
3.) Run Thales DDC Report to export a list of datasources with sensitive data.
4.) If any of the sources are of file types excel,word,pdf run sample code ExtractFilesFromDDCReport.java to generate a list of the datsources that need to be converted to text.
5.) Run sample code ConvertToTextBatchProcessor.java to convert files found in step for to text version.
6.) Depending on customer preference for CSP (AWS,GCP,Azure) run appropriate sample code ThalesCSPProtectRevealBatchProcessor.java to find sensitive fields in datasources and create a new output with sensitive data protected.
7.) Now load protected data into appropriate AWS,GCP, Azure training data upload.

Thales Protect/Reveal Options

The solutions provide in this document for field level protection are based on two application development offerings from Thales CRDP and CADP for Java.

1.)CRDP

CipherTrust RESTful Data Protection (CRDP) is a RESTful webservice that protects sensitive data using encryption, tokenization, or data generalization. CRDP is designed from the ground up to seamlessly fit with existing cloud and on-premise applications. CRDP is deployed as a container and performs a wide

range of data protection functions on the sensitive data including data masking, pseudonymization, and redaction. CRDP is easy to deploy, configure, manage, and migrate.
https://thalesdocs.com/ctp/con/crdp/latest/index.html

2.) CipherTrust Application Data Protection for Java (CADP for Java)

Is a Java Cryptography Extension (JCE) provider that enables users to integrate the Key Manager's capabilities into their Java applications. CADP for Java is available in two variants:

- **Centrally Managed APIs**
- **Traditional APIs**

The examples provided in this document are based on Centrally Managed APIs.

This is our new set of APIs available from CADP for Java 8.18.1 release. CADP for Java offers simplified APIs (Protect/Reveal) to perform cryptography operations. CADP for Java integrates seamlessly with **Application Data Protection** on CipherTrust Manager, enabling organizations to centrally configure, manage, and enforce data-centric cryptographic policies in a reusable, human-readable format, providing flexibility and ease of management.
Protection policy (ciphers, keys, IV, tweak, and so on) defines how to protect the sensitive data. Whereas, Access policy determines who can view sensitive data and how (plaintext, ciphertext, custom masking format or redacted).
Centrally Managed APIs offer several advantages over traditional APIs, including:

- Developers do not need to understand cryptographic parameters (cipher, key, IV, Nonce, Tweak, and so on) to protect data as the Data Security Admins are responsible to handle configurations and policies.
- Each deployed application with CADP for Java is visible on CipherTrust Manager (providing a Single Pane of Glass view).
- Data Security Admins gain Crypto Agility, enabling real time changes to cipher, keys, and parameters.

https://thalesdocs.com/ctp/con/cadp/cadp-java/latest/admin/index.html#centrally-managed-apis

The code provided in this document provides examples using both the CRDP and CADP for Java Centralized API's. As you can see from the code there are two helper classes that make it easy to switch between either api.

```
// Initialize ThalesProtectRevealHelper. Two options are commented out, showing different
// implementations (CADP vs. REST). The REST implementation is currently active.
// ThalesProtectRevealHelper tprh = new ThalesCADPProtectRevealHelper(keymanagerhost, crdptkn,
null, policyType,showmeta);

// Using ThalesRestProtectRevealHelper for content protection and revelation.
ThalesProtectRevealHelper tprh = new ThalesRestProtectRevealHelper(crdpip, metadata,
policyType,showmeta);

Note: For each option there will be different properties required please refer to the code
samples for detailed explanations.
```