10
Cloud Data Loss Prevention

In this chapter, we will look at Google Cloud data loss protection products and capabilities. **Data Loss Prevention** (**DLP**) is a strategy for detecting and preventing the exposure and exfiltration of sensitive data. Google's DLP strategy involves a layered approach. In addition to a proper organizational hierarchy, network security, IAM access, and **VPC Service Controls** (**VPC-SC**), DLP plays a key role in data protection.

Cloud DLP is quite widely used in data pipelines, especially for data warehouses. Protecting confidential data is one of the critical aspects of data workloads, so Cloud DLP helps customers gain visibility of sensitive data risks across the organization. We will look at several features of Cloud DLP, how to configure the product to do inspection and de-identification, and some best practices. There are few tutorials in the chapter, so try out examples to get a solid understanding.

In this chapter, we will cover the following topics:

- Overview of Cloud DLP
- DLP architecture
- How DLP discovery works
- De/re-identification of data
- How to create a DLP scan job
- DLP use cases
- How to mask and tokenize sensitive data
- Best practices and design considerations
- Data exfiltration controls

Overview of Cloud DLP

Cloud DLP offers some key features for Google's customers:

- BigQuery-based data warehouses can be profiled to detect sensitive data, allowing for automated sensitive data discovery. You can scan through the entire Google Cloud organization or choose folders or projects with the profiler's flexibility.
- Over 150 built-in information detectors are available in Cloud DLP. Because DLP is API-based, it can be used to swiftly scan, discover, and classify data from anywhere.
- Support for Cloud Storage, Datastore, and BigQuery is built in: Cloud DLP comes with built-in support for these storage options.
- You can calculate the level of risk to your data privacy: quasi-identifiers are data elements or combinations of data that can be linked to a single individual or a small group of people. Cloud DLP gives you the option to examine statistical features such as k-anonymity and l-diversity, allowing you to assess the risk of data re-identification.
- DLP classification results can be delivered straight to BigQuery for more thorough analysis or exported to your analytical instrument of choice. BigQuery and Data Studio can be used to create bespoke reports.
- Cloud DLP secures your data by storing it in memory. The data in the backend is not persistent. In addition, the product is subjected to various independent third-party audits to ensure data confidentiality, privacy, and safety.
- Cloud DLP may be swiftly deployed using reusable templates, with periodic scans monitoring the data and Pub/Sub notifications for integration into serverless architectures.
- To suit your company's needs, you can add your own custom info types, alter detection levels, and establish detection rules.

Now that you understand Cloud DLP's capabilities at a high level, let us understand the DLP architecture.

DLP architecture options

Cloud DLP primarily is seen in three architecture patterns: the content/streaming and storage methods and a hybrid architecture that combines these two patterns.

Content methods

In this architecture option, the data is streamed to the Cloud DLP APIs for inspection/classification or de-identification/transformation. A synchronous API response is received from Cloud DLP. In this case, the client application is expected to process the response. This architecture is typically seen in data pipelines or call center applications where real-time response is needed.

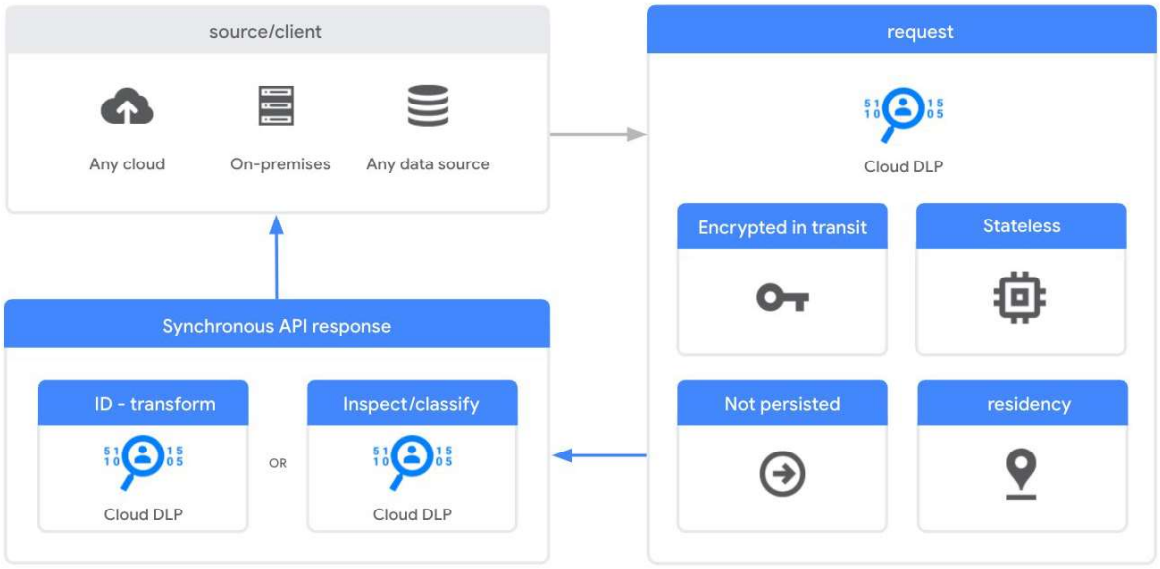

Figure 10.1 – Content method architecture

As shown in *Figure 10.1*, using content inspection, you stream small payloads of data to Cloud DLP along with instructions about what to inspect for. Cloud DLP then inspects the data for sensitive content and **personally identifiable information** (**PII**) and returns the results of its scan back to you.

Storage methods

In this architecture option, a job is set up based on a trigger to scan for sensitive data. The results of the scans are published so that action can be taken based on the results. The results can be pushed to **Security Operations Center** (**SOC**) tools such as Security Command Center so security responders can take action or an automated response detection can be built using Cloud Functions or a more sophisticated product such as SOAR.

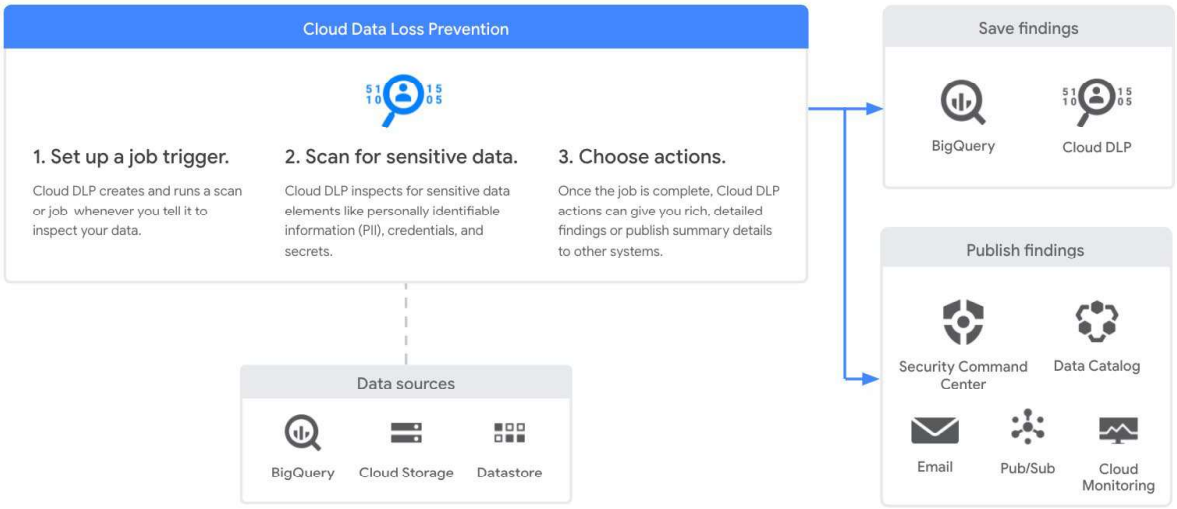

Figure 10.2 – Storage method architecture

As shown in *Figure 10.2*, in this architecture option, you tell Cloud DLP what to inspect and then Cloud DLP runs a job that scans the repository. After the scan is complete, Cloud DLP saves a summary of the results of the scan back to the job. You can additionally specify that the results are sent to another Google Cloud product for analysis, such as a separate BigQuery table or Security Command Center.

The type of scan typically involves those Google Cloud projects where you do not expect to have sensitive data. For projects that are meant to have sensitive data such as cardholder data or PII, you should also invest in data exfiltration controls such as VPC Service Controls for DLP.

Hybrid methods

Hybrid jobs and job triggers enable you to broaden the scope of protection that Cloud DLP provides beyond simple content inspection requests and **Google Cloud Storage** (**GCS**) repository scanning.

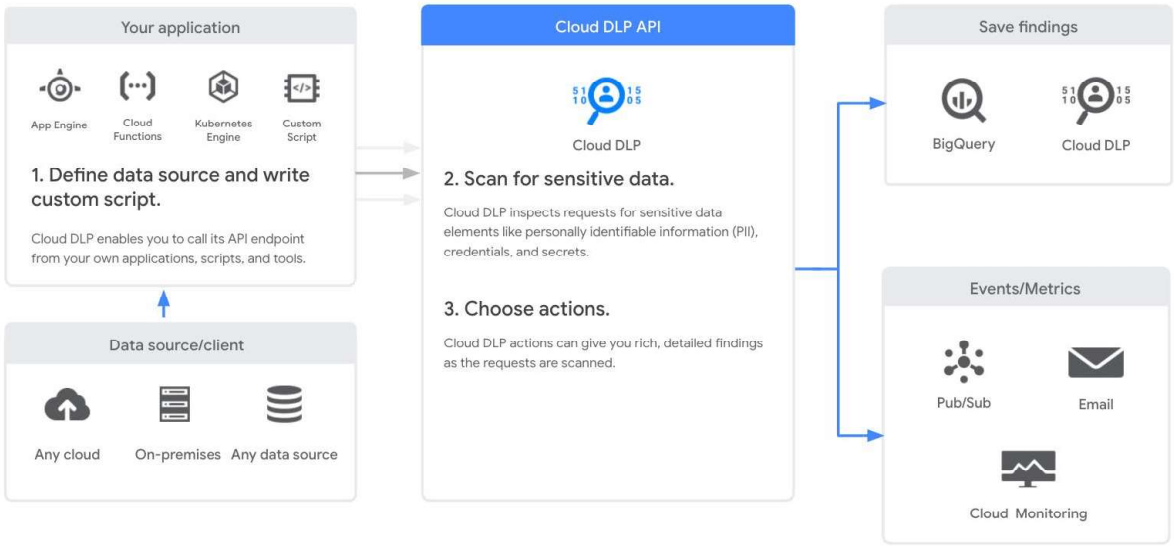

Figure 10.3 – Hybrid inspection architecture

As shown in *Figure 10.3*, you may feed data from practically any source—including sources outside Google Cloud—directly to Cloud DLP via hybrid jobs and job triggers, and Cloud DLP will analyze the data for sensitive information and automatically record and aggregate the scan results for further analysis.

Hybrid jobs and *job triggers* encompass a set of asynchronous API methods that allow you to scan payloads of data sent from virtually any source for sensitive information, and then store the findings in Google Cloud. Hybrid jobs enable you to write your own data crawlers that behave and serve data similarly to the Cloud DLP storage inspection methods.

Hybrid environments are common in enterprises. Many organizations store and process sensitive data using some combination of the following:

- Other cloud providers
- On-premises servers or other data repositories
- Non-native storage systems, such as systems running inside a virtual machine
- Web and mobile apps
- Google Cloud-based solutions

Using hybrid jobs, Cloud DLP can inspect data sent to it from any of these sources. Example scenarios include the following:

- Inspect data stored in Amazon **Relational Database Service** (**RDS**), MySQL running inside a virtual machine, or an on-premises database
- Inspect and tokenize data as you migrate from on-premises to the cloud, or between production, development, and analytics
- Inspect and de-identify transactions from a web or mobile application before storing the data at rest

The basic workflow for using hybrid jobs and job triggers is as follows:

1. You write a script or create a workflow that sends data to Cloud DLP for inspection along with some metadata.
2. You configure and create a hybrid job resource or trigger and enable it to activate when it receives data.
3. Your script or workflow runs on the client side and sends data to Cloud DLP. The data includes an activation message and the job trigger's identifier, which triggers the inspection.
4. Cloud DLP inspects the data according to the criteria you set in the hybrid job or trigger.
5. Cloud DLP saves the results of the scan to the hybrid job resource, along with the metadata that you provide. You can examine the results using the Cloud DLP UI in the Cloud console.
6. Optionally, Cloud DLP can run post-scan actions, such as saving inspection results data to a BigQuery table or notifying you by email or Pub/Sub.

A hybrid job trigger enables you to create, activate, and stop jobs so that you can trigger actions whenever you need them. By ensuring that your script or code sends data that includes the hybrid job trigger's identifier, you don't need to update your script or code whenever a new job is started.

Now that you understand the different inspection methods, let us get started with some foundational aspects of DLP.

Cloud DLP terminology

Before we jump into defining Cloud DLP inspection templates, let us go over some important terminology that you will see in the templates.

DLP infoTypes

Information types, also known as infoTypes, are sensitive data kinds that Cloud DLP is preconfigured to scan and identify—for instance, US Social Security numbers, credit card numbers, phone numbers, zip codes, and names. Both built-in and custom InfoTypes are supported by Cloud DLP.

There is a detector for each infoType defined in Cloud DLP. To identify what to look for and how to transform findings, Cloud DLP employs infoType detectors in its scan configuration. When showing or reporting scan findings, infoType names are also used. Cloud DLP releases new infoType detectors and groups regularly. Call the Cloud DLP REST API's `infoTypes.list` method to receive the most up-to-date list of built-in infoTypes.

Please keep in mind that the built-in infoType detectors aren't always reliable. Google suggests that you evaluate your settings to ensure that they comply with your regulatory requirements.

Here are some examples of infoType detectors (this is not an exhaustive list):

- `ADVERTISING_ID`: Identifiers used by developers to track users for advertising purposes. These include Google Play **Advertising IDs**, Amazon **Advertising IDs**, Apple's **Identifier For Advertising** (**IDFA**), and Apple's **Identifier For Vendor** (**IDFV**).
- `AGE`: An age measured in months or years.
- `CREDIT_CARD_NUMBER`: A credit card number is 12 to 19 digits long. They are used for payment transactions globally.
- `CREDIT_CARD_TRACK_NUMBER`: A credit card track number is a variable-length alphanumeric string. It is used to store key cardholder information.
- `DATE`: A date. This infoType includes most date formats, including the names of common world holidays.
- `DATE_OF_BIRTH`: A date that is identified by context as a date of birth. Note that this is not recommended for use during latency-sensitive operations.
- `DOMAIN_NAME`: A domain name as defined by the DNS standard.
- `EMAIL_ADDRESS`: An email address identifies the mailbox that emails are sent to or from. The maximum length of the domain name is 255 characters, and the maximum length of the local part is 64 characters.
- `US_SOCIAL_SECURITY_NUMBER`: A United States **Social Security number** (**SSN**) is a nine-digit number issued to US citizens, permanent residents, and temporary residents. This detector will not match against numbers with all zeros in any digit group (that is, `000-##-####`, `###-00-####`, or `###-##-0000`), against numbers with 666 in the first digit group, or against numbers whose first digit is nine.

Now we have seen the overview of the Cloud DLP info types, let us understand the technical terms that you will frequently come across while working with Cloud DLP.

Data de-identification

De-identification is a term used to represent a process that removes personal information from a dataset. You will also come across the term *redaction* in relation to Cloud DLP. Redaction in general refers to the removal or masking of information, while de-identification refers to a process of changing it so you no longer identify the original text or are able to derive meaning from it. De-identified data is considered suitable for applications since it doesn't contain any PII that can be used to trace the results back to the user. There are several ways data can be de-identified:

- Masking sensitive data by partially or fully replacing characters with a symbol, such as an asterisk (*) or hash (#)
- Replacing each instance of sensitive data with a token, or surrogate, string
- Encrypting and replacing sensitive data using a randomly generated or pre-determined key

Cloud DLP supports the following methods of de-identification:

- **Masking**: Replaces the original data with a specified character, either partially or completely.
- **Replacement**: Replaces original data with a token or the name of the infoType if detected.
- **Date shifting**: Date-shifting techniques randomly shift a set of dates but preserve the sequence and duration of a period of time. Shifting dates is usually done in relation to an individual or an entity. That is, each individual's dates are shifted by an amount of time that is unique to that individual.
- **Generalization and bucketing**: Generalization is the process of taking a distinguishing value and abstracting it into a more general, less distinguishing value. Generalization attempts to preserve data utility while also reducing the identifiability of the data. One common generalization technique that Cloud DLP supports is bucketing. With bucketing, you group records into smaller buckets in an attempt to minimize the risk of an attacker associating sensitive information with identifying information. Doing so can retain meaning and utility, but it will also obscure the individual values that have too few participants.
- **Pseudonymization**: This is a de-identification technique that replaces sensitive data values with cryptographically generated tokens:
    - **Two-way tokenization pseudonymization**: Replaces the original data with a token that is deterministic, preserving referential integrity. You can use the token to join data or use the token in aggregate analysis. You can reverse or de-tokenize the data using the same key that you used to create the token. There are two methods for two-way tokenization:

- **Deterministic Encryption (DE) using AES-SIV: Advanced Encryption Standard-Synthetic Initialization Vector (AES-SIV)** is a cryptographic mode of operation designed to provide secure and authenticated encryption of data. It is based on the **Advanced Encryption Standard (AES)** block cipher and is intended for applications that require both data confidentiality and data integrity. It is commonly used for protecting data stored in databases, files, or other forms of persistent storage. AES-SIV also provides a way to protect data against replay attacks and can be used to construct secure communication protocols. Using AES-SIV, an input value is replaced with a value that has been encrypted using the AES-SIV encryption algorithm with a cryptographic key, encoded using base64, and then prepended with a surrogate annotation, if specified. This method produces a hashed value, so it does not preserve the character set or the length of the input value. Encrypted, hashed values can be re-identified using the original cryptographic key and the entire output value, including surrogate annotation. Learn more about the format of values tokenized using AES-SIV encryption.
- **Format Preserving Encryption (FPE) with Flexible Format-Preserving Encryption (FFX)**: This is an encryption technique that encrypts data while preserving the format of the original data. This means that FFX encryption can be used to encrypt data such as credit card numbers, SSNs, and other sensitive data without changing the format of the original data. FFX is used to protect data while it is stored or transmitted and to prevent unauthorized access to the data. Using FPE-FFX, an input value is replaced with a value that has been encrypted using the FPE-FFX encryption algorithm with a cryptographic key, and then prepended with a surrogate annotation, if specified. By design, both the character set and the length of the input value are preserved in the output value. Encrypted values can be re-identified using the original cryptographic key and the entire output value, including the surrogate annotation.

- **One-way tokenization using cryptographic hashing pseudonymization**: An input value is replaced with a value that has been encrypted and hashed using **Hash-Based Message Authentication Code Secure Hash Algorithm** 256 or **HMAC-SHA56** on the input value with a cryptographic key. The hashed output of the transformation is always the same length and can't be re-identified. Learn more about the format of values tokenized using cryptographic hashing.

Refer to the following table for a comparison of these three pseudonymization methods.

	Deterministic encryption using AES-SIV	**Format preserving encyption**	**Cryptographic hashing**
Encryption type	AES-SIV	FPR-FFX	HMAC-SHA256
Supported input values	At least 1 char long; no character set limitations.	At least 2 chats long; must be encoded as ASCII.	Must be a string or an integer value.
Surrogate annotation	Optional	Optional	N/A
Context tweak	Optional	Optional	N/A
Character set and length preserved	No	Yes	No
Reversible	Yes	Yes	No
Referential integrity	Yes	Yes	Yes

Table 10.1 – Comparison of the pseudonymization methods

Method selection

Choosing the best de-identification method can vary based on your use case. For example, if a legacy app is processing the de-identified records, then format preservation might be important. If you're dealing with strictly formatted 10-digit numbers, FPE preserves the length (10 digits) and character set (numeric) of an input for legacy system support.

However, if strict formatting isn't required for legacy compatibility, as is the case for values in the cardholder's name column, then DE is the preferred choice because it has a stronger authentication method. Both FPE and DE enable the tokens to be reversed or de-tokenized. If you don't need de-tokenization, then cryptographic hashing provides integrity but the tokens can't be reversed.

Other methods—such as masking, bucketing, date-shifting, and replacement—are good for values that don't need to retain full integrity. For example, bucketing an age value (for example, 27) to an age range (20-30) can still be analyzed while reducing the uniqueness that might lead to the identification of an individual.

Token encryption keys

A cryptographic key, also known as a token encryption key, is necessary for cryptographic de-identification transformations. The same token encryption key that is used to de-identify the original value is also used to re-identify it. This book does not cover the secure development and maintenance of token encryption keys.

However, there are a few key aspects to keep in mind that will be applied later in the lessons:

- In the template, avoid using plaintext keys. Instead, build a wrapped key with Cloud KMS.
- To limit the danger of keys being compromised, use different token encryption keys for each data element.
- Token encryption keys should be rotated. Although the wrapped key can be rotated, the token encryption key cannot be rotated because it compromises the tokenization's integrity. You must re-tokenize the entire dataset when the key is rotated.

Now that we are familiar with the basics of de-identification and which method is better suited for our use case, let us walk through how to create an inspection template. Later we will look at how to use de-identification.

Remember that Cloud DLP requires an inspection of the data before de-identification can be performed. The Cloud DLP inspection process identifies the infoTypes that further can be used in de-identification. The inspection step is *not* optional.

Creating a Cloud DLP inspection template

The first step in using classification capabilities is to create an inspection template. The inspection template will store all the data classification requirements:

1. In the Cloud console, open **Cloud DLP**.
2. From the **CREATE** menu, choose **Template**.

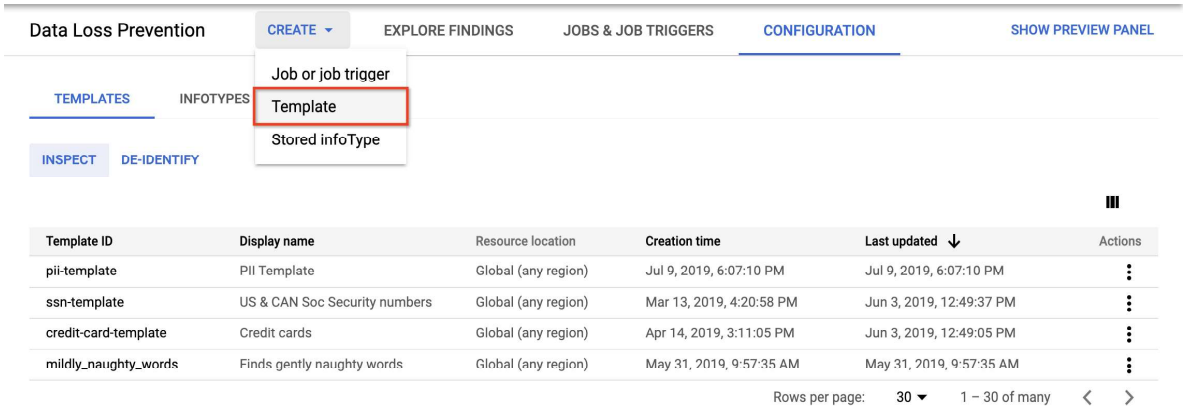

Figure 10.4 – Creating a DLP inspection template

3. Alternatively, click the following button: **Create new template**.

This page contains the following sections:

- **Define template**
- **Configure detection**

Defining the template

Under **Define template**, enter an identifier for the inspection template. This is how you'll refer to the template when you run a job, create a job trigger, and so on. You can use letters, numbers, and hyphens. If you want, you can also enter a more human-friendly display name, as well as a description to better remember what the template does.

Configuring detection

Next, you configure what Cloud DLP detects in your content by choosing an infoType and other options.

Under **InfoTypes**, choose the infoType detectors that correspond to a data type you want to scan for. You can also leave this field blank to scan for all default infoTypes. More information about each detector is provided in the *Further reading* section at the end of this chapter.

You can also add custom infoType detectors in the **Custom infoTypes** section, and customize both built-in and custom infoType detectors in the **Inspection rulesets** section.

Custom infoTypes

Be aware that any custom infoType detector you create here is specific to this workflow and can't be reused elsewhere. The one exception is *Stored infoType*, which requires that you create the stored custom infoType detector before specifying it here.

To add a custom infoType detector, do the following:

1. Click **Add custom infoType**.
2. Choose the type of custom infoType detector you want to create:
    - **Words or phrases**: Matches on one or more words or phrases that you enter into the field. Use this custom infoType when you have just a few words or phrases to search for. Give your custom infoType a name, and then type the word or phrase you want Cloud DLP to match. To search for multiple words or phrases, press *Enter* after each one.
    - **Dictionary path**: Searches your content for items in a list of words and phrases. The list is stored in a text file in Cloud Storage. Use this custom infoType when you have anywhere from a few to several hundred thousand words or phrases to search for. This method is also useful if your list contains sensitive elements and you don't want to store them inside of a job or template. Give your custom infoType a name, and then, under **Dictionary location**, enter or browse to the Cloud Storage path where the dictionary file is stored.
    - **Regular expression** (**regex**): Matches content based on a regular expression. Give your custom infoType a name, and then, in the **Regex** field, enter a regex pattern to match words and phrases. See the supported regex syntax in Google Cloud documentation at `https://packt.link/dRR06`.
    - **Stored infoType**: This option adds a stored custom dictionary detector, which is a kind of dictionary detector that is built from either a large text file stored in Cloud Storage or a single column of a BigQuery table. Use this kind of custom infoType when you have anywhere from several hundred thousand to tens of millions of words or phrases to search for. Be aware that this is the only option in this menu for which you must have already created the stored infoType to use it. Give your custom infoType a name (different from the name you gave the stored infoType), and then, in the **Stored infoType** field, enter the name of the stored infoType.
3. Click **Add custom infoType** again to add additional custom infoType detectors.

Now let us see how to use rulesets to customize the infoTypes to match your requirements. This helps you to avoid false positives.

Inspection rulesets

Inspection rulesets allow you to customize both built-in and custom infoType detectors using context rules. The two types of inspection rules are as follows:

- **Exclusion rules**, which help exclude false or unwanted findings
- **Hotword rules**, which help adjust the likelihood of the finding based on how near the hotword is to the finding

To add a new ruleset, first, specify one or more built-in or custom infoType detectors in the infoTypes section. These are the infoType detectors that your rulesets will be modifying. Then, do the following:

1. Click in the **Choose infoTypes** field. The infoType or infoTypes you specified previously will appear below the field in a menu, as shown in *Figure 10.5*.

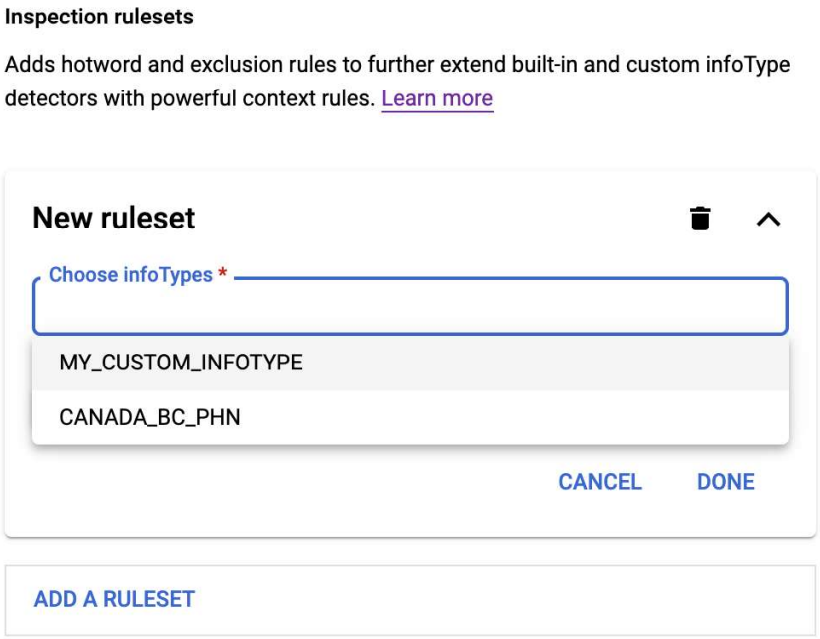

Figure 10.5 – Choosing infoTypes for Inspection rulesets

2. Choose an infoType from the menu, and then click **Add rule**. A menu appears with two options: **Hotword rule** and **Exclusion rule**.

   For hotword rules, choose **Hotword rule**. Then, do the following:

   I. In the **Hotword** field, enter a regular expression that Cloud DLP should look for.

   II. From the **Hotword proximity** menu, choose whether the hotword you entered is found before or after the chosen infoType.

   III. In **Hotword distance from infoType**, enter the approximate number of characters between the hotword and the chosen infoType.

   IV. In **Confidence level adjustment**, choose whether to assign matches to a fixed likelihood level or to increase or decrease the default likelihood level by a certain amount.

   For exclusion rules, choose **Exclusion rules**. Then, do the following:

   V. In the **Exclude** field, enter a regular expression (regex) that Cloud DLP should look for.

   VI. From the **Matching type** menu, choose one of the following:

   - **Full match**: The finding must completely match the regex.
   - **Partial match**: A substring of the finding can match the regex.
   - **Inverse match**: The finding doesn't match the regex.

You can add additional hotword or exclusion rules and rulesets to further refine your scan results.

Confidence threshold

Every time Cloud DLP detects a potential match for sensitive data, it assigns it a likelihood value on a scale from `Very unlikely` to `Very likely`. When you set a likelihood value here, you are instructing Cloud DLP to only match data that corresponds to that likelihood value or higher.

The default value of `Possible` is sufficient for most purposes. If you routinely get matches that are too broad, move the slider up. If you get too few matches, move the slider down.

When you're done, click **Create** to create the template. The template's summary information page appears.

Let us discuss some of the best practices for inspection.

Best practices for inspecting sensitive data

There are several things that you need to consider before starting an inspection. We will go over them now:

- **Identify and prioritize scanning**: It's important to identify your resources and specify which have the highest priority for scanning. When just getting started, you may have a large backlog of data that needs classification, and it'll be impossible to scan it all immediately. Choose data initially that poses the highest risk—for example, data that is frequently accessed, widely accessible, or unknown.
- **Reduce latency**: Latency is affected by several factors: the amount of data to scan, the storage repository being scanned, and the type and number of infoTypes that are enabled. To help reduce job latency, you can try the following:
  - Enable sampling.
  - Avoid enabling infoTypes you don't need. While useful in certain scenarios, some infoTypes—including `PERSON_NAME`, `FEMALE_NAME`, `MALE_NAME`, `FIRST_NAME`, `LAST_NAME`, `DATE_OF_BIRTH`, `LOCATION`, `STREET_ADDRESS`, and `ORGANIZATION_NAME`—can make requests run much more slowly than requests that do not include them.
  - Always specify infoTypes explicitly. Do not use an empty infoTypes list.
  - Consider organizing the data to be inspected into a table with rows and columns, if possible, to reduce network round trips.
- **Limit the scope of your first scans**: For the best results, limit the scope of your first scans instead of scanning all of your data. Start with a few requests. Your findings will be more meaningful when you fine-tune what detectors to enable and what exclusion rules might be needed to reduce false positives. Avoid turning on all infoTypes if you don't need them all, as false positives or unusable findings may make it harder to assess your risk. While useful in certain scenarios, some infoTypes such as `DATE`, `TIME`, `DOMAIN_NAME`, and `URL` detect a broad range of findings and may not be useful to turn on.
- **Limit the amount of content inspected**: If you are scanning BigQuery tables or Cloud Storage buckets, Cloud DLP includes a way to scan a subset of the dataset. This has the effect of providing a sampling of scan results without incurring the potential costs of scanning an entire dataset.

> **Note**
>
> Due to the need to scan the entire image for sensitive data, sampling is not supported for image file types. Any images scanned will be billed according to the size of the image file.

- **Inspect data on-premises or in other clouds**: If the data to be scanned resides *on-premises or outside of Google Cloud*, use the API methods `content.inspect` and `content.deidentify` to scan the content to classify findings and pseudonymized content without persisting the content outside of your local storage.

We have seen how to inspect data and best practices around data inspection. Let us now understand how to de-identify sensitive data.

Inspecting and de-identifying PII data

To de-identify sensitive data, use Cloud DLP's `content.deidentify` method.

There are three parts to a de-identification API call:

- **The data to inspect**: A string or table structure (`ContentItem` object) for the API to inspect.
- **What to inspect for**: Detection configuration information (`InspectConfig`) such as what types of data (or infoTypes) to look for, whether to filter findings that are above a certain likelihood threshold, whether to return no more than a certain number of results, and so on. Not specifying at least one infoType in an `InspectConfig` argument is equivalent to specifying all built-in infoTypes. Doing so is not recommended, as it can cause decreased performance and increased cost.
- **What to do with the inspection findings**: Configuration information (`DeidentifyConfig`) that defines how you want the sensitive data de-identified. This argument is covered in more detail in the following section.

The API returns the same items you gave it, in the same format, but any text identified as containing sensitive information according to your criteria is *de-identified*. Now let us look at various de-identification transformations.

De-identification transformations

We saw how to inspect data but many times you want to inspect and de-identify the data. Now we will see how to do that using various transformations supported by DLP. You must specify one or more transformations when you set the de-identification configuration (`DeidentifyConfig`). There are two categories of transformations:

- **InfoTypeTransformations**: Transformations that are only applied to values within the submitted text that are identified as a specific infoType.
- **RecordTransformations**: Transformations that are only applied to values within submitted tabular text data that are identified as a specific infoType, or on an entire column of tabular data.

Now let us go over each deidentification configuration.

replaceConfig

`replaceConfig` will replace any sensitive data with a string you specify.

redactConfig

`redactConfig` redacts a given value by removing it completely.

characterMaskConfig

Setting `characterMaskConfig` to a `CharacterMaskConfig` object partially masks a string by replacing a given number of characters with a fixed character. Masking can start from the beginning or end of the string.

cryptoHashConfig

Setting `cryptoHashConfig` to a `CryptoHashConfig` object performs pseudonymization on an input value by generating a surrogate value using cryptographic hashing.

This method replaces the input value with an encrypted *digest*, or hash value. The digest is computed by taking the SHA-256 hash of the input value. The method outputs a base64-encoded representation of the hashed output. Currently, only string and integer values can be hashed.

dateShiftConfig

Setting `dateShiftConfig` to a `DateShiftConfig` object performs date shifting on a date input value by shifting the dates by a random number of days.

Date-shifting techniques randomly shift a set of dates but preserve the sequence and duration of a period of time. Shifting dates is usually done in relation to an individual or an entity. You might want to shift all of the dates for a specific individual using the same shift differential but use a separate shift differential for each other individual.

Now that you have understood different methods of de-identification, let us go over a tutorial on how to do this in practice. Make sure you have a Google Cloud project ready to execute the steps covered next.

Tutorial: How to de-identify and tokenize sensitive data

Cloud DLP supports both reversible and non-reversible cryptographic methods. In order to re-identify content, you need to choose a reversible method. The cryptographic method described here is called deterministic encryption using **Advanced Encryption Standard in Synthetic Initialization Vector mode** (**AES-SIV**). We recommend this among all the reversible cryptographic methods that Cloud DLP supports because it provides the highest level of security.

In this tutorial, we're going to see how to generate a key to de-identify sensitive text into a cryptographic token. In order to restore (re-identify) that text, you need the cryptographic key that you used during de-identification and the token.

Before you begin, make sure you have the following roles in your Google Cloud project:

- Service account admin, to be able to create service accounts
- Service usage admin, to be able to enable services
- Security admin, to be able to grant roles

Once you have the right roles, follow these steps. The steps walk you through the process of creating an AES key to be able to de-identify sensitive text into a cryptographic token:

1. In the Google Cloud console, on the project selector page, select or create a new Google Cloud project.
2. Make sure that billing is enabled for your cloud project.
3. Enable the Cloud DLP and Cloud KMS APIs.
4. Create a service account by following the next steps:

    I. In the Cloud console, navigate to the **Create service account** page.

    II. In the **Service account name** field, enter a name. The Cloud console fills in the **Service account ID** field based on this name. In the **Service account description** field, enter a description.

    III. Click **Create** and continue.

    IV. To provide access to your project, grant the following role(s) to your service account: **Project > DLP Administrator**.

> **Note**
> In production environments, do not grant the Owner, Editor, or Viewer roles. Instead, grant a predefined role or custom role that meets your needs.

    V. Click **Continue**.

    VI. Click **Done** to finish creating the service account.

    Do not close your browser window. You will use it in the next step.

5. Create a service account key:

    I. In the Cloud console, click the email address for the service account that you just created.

    II. Click **Keys**.

    III. Click **Add key**, then click **Create new key**.

    IV. Click **Create**. A JSON key file is downloaded to your computer.

> **Note**
> It is against security practice to create a key and keep it forever. You should delete this key as soon as this tutorial is finished.

    V. Click **Close**.

6. Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the path of the JSON file that contains your service account key. This variable only applies to your current shell session, so if you open a new session, set the variable again.

Now that you have the right service account, we will create a KMS key that will be used to wrap the AES key used for the actual encryption of the DLP token.

Step 1: Creating a key ring and a key

Before you start this procedure, decide where you want Cloud DLP to process your de-identification and re-identification requests. When you create a Cloud KMS key, you must store it either globally or in the same region that you will use for your Cloud DLP requests. Otherwise, the Cloud DLP requests will fail.

You can find a list of supported locations in Cloud DLP locations. Take note of the name of your chosen region (for example, `us-west1`).

This procedure uses global as the location for all API requests. If you want to use a different region, replace global with the region name.

Create a key ring:

```
gcloud kms keyrings create "dlp-keyring" --location "global"
```

Create a key:

```
gcloud kms keys create "dlp-key" --location "global"  --keyring
"dlp-keyring" --purpose "encryption
```

List your key ring and key:

```
gcloud kms keys list --location "global" --keyring "dlp-keyring"
```

You will get the following output:

```
NAME: projects/PROJECT_ID/locations/global/keyRings/dlp-keyring/
cryptoKeys/dlp-key
PURPOSE: ENCRYPT_DECRYPT
ALGORITHM: GOOGLE_SYMMETRIC_ENCRYPTION
PROTECTION_LEVEL: SOFTWARE
LABELS:
PRIMARY_ID: 1
PRIMARY_STATE: ENABLED
```

In this output, `PROJECT_ID` is the ID of your project.

The path under `NAME` is the full resource name of your Cloud KMS key. Take note of it because the de-identify and re-identify requests require it.

Step 2: Creating a base64-encoded AES key

This section describes how to create an **Advanced Encryption Standard** (**AES**) key and encode it in base64 format. This key shall be used to encrypt the actual sensitive data. As you can see, you have complete control over the generation and maintenance of this key.

> **Note**
>
> These steps use the `openssl` and `base64` commands, but there are a few other ways to perform this task based on your security policies.

Now use the following command to create a 256-bit key in the current directory:

```
openssl rand -out "./aes_key.bin" 32
```

The `aes_key.bin` file is added to your current directory.

Encode the AES key as a `base64` string:

```
base64 -i ./aes_key.bin
```

You will get an output similar to the following:

```
uEDo6/yKx+zCg2cZ1DBwpwvzMVNk/c+jWs7OwpkMc/s=
```

> **Warning**
> Do not use this example key to protect actual sensitive workloads. This key is provided only to serve as an example. Because it's shared here, this key is not safe to use.

Step 3: Wrapping the AES key using the Cloud KMS key

This section describes how to use the Cloud KMS key that you created in *Step 1* to wrap the base64-encoded AES key that you created in *Step 2: Creating a base64-encoded AES key*.

To wrap the AES key, use `curl` to send the following request to the Cloud KMS API `projects.locations.keyRings.cryptoKeys.encrypt`:

```
curl "https://cloudkms.googleapis.com/v1/projects/PROJECT_ID/
locations/global/keyRings/dlp-keyring/cryptoKeys/dlp-key:encrypt"
--request "POST"
  --header "Authorization:Bearer $(gcloud auth application-default
print-access-token)" --header "content-type: application/json" --data
"{\"plaintext\": \"BASE64_ENCODED_AES_KEY\"}"
```

Replace the following:

- `PROJECT_ID`: The ID of your project.
- `BASE64_ENCODED_AES_KEY`: The base64-encoded string returned in *Step 2: Creating a base64-encoded AES key*.

The response that you get from Cloud KMS is similar to the following JSON:

```
{
  "name": "projects/PROJECT_ID/locations/global/keyRings/dlp-keyring/
cryptoKeys/dlp-key/cryptoKeyVersions/1",
  "ciphertext":
"CiQAYuuIGo5DVaqdE0YLioWxEhC8LbTmq7Uy2G3qOJlZB7WXBw0SSQAj
dwP8ZusZJ3Kr8GD9W0vaFPMDksmHEo6nTDaW/
j5sSYpHa1ym2JHk+lUgkC3Zw5bXhfCNOkpXUdHGZKou1893O8BDby/82HY=",
  "ciphertextCrc32c": "901327763",
  "protectionLevel": "SOFTWARE"
}
```

In this output, `PROJECT_ID` is the ID of your project.

Take note of the value of ciphertext in the response that you get. That is your wrapped AES key.

Step 4: Sending a de-identify request to the Cloud DLP API

This section describes how to de-identify sensitive data in text content.

To complete this task, you need the following:

- The full resource name of the Cloud KMS key that you created in *Step 1: Creating a key ring and a key*
- The wrapped key that you created in *Step 3: Wrapping the AES key using the Cloud KMS key*

To de-identify sensitive data in text content, follow these steps:

1. Create a JSON request file with the following text:

```
{
  "item": {
    "value": "My name is Alicia Abernathy, and my email address
is aabernathy@example.com."
  },
  "deidentifyConfig": {
    "infoTypeTransformations": {
      "transformations": [
        {
          "infoTypes": [
            {
              "name": "EMAIL_ADDRESS"
            }
          ],
          "primitiveTransformation": {
            "cryptoDeterministicConfig": {
              "cryptoKey": {
                "kmsWrapped": {
                  "cryptoKeyName": "projects/PROJECT_ID/
locations/global/keyRings/dlp-keyring/cryptoKeys/dlp-key",
                  "wrappedKey": "WRAPPED_KEY"
                }
              },
              "surrogateInfoType": {
                "name": "EMAIL_ADDRESS_TOKEN"
              }
            }
          }
        }
      ]
    }
```

```
    },
    "inspectConfig": {
      "infoTypes": [
        {
          "name": "EMAIL_ADDRESS"
        }
      ]
    }
  }
```

2. Replace the following:

- `PROJECT_ID`: The ID of your project.
- `WRAPPED_KEY`: The wrapped key that you created in *Step 3: Wrapping the AES key using the Cloud KMS key*.

  Make sure that the resulting value of `cryptoKeyName` forms the full resource name of your Cloud KMS key.

3. Save the file as `deidentify-request.json`.

Step 5: Sending a de-identity request to the Cloud DLP API

Now let us send the de-identification request to the DLP API based on the file you created in the previous step:

```
curl -s -H "Authorization: Bearer $(gcloud auth application-
default print-access-token)" -H "Content-Type: application/json"
https://dlp.googleapis.com/v2/projects/dlp-tesing/locations/global/
content:deidentify -d @deidentify-request.json
```

You should see output similar to this:

```
{
  "item": {
    "value": "My name is Alicia Abernathy, and my email
address is EMAIL_ADDRESS_TOKEN(52):ARa5jvGRxjop/UOzU9DZQAlCT/
yoTOjcOws7I/2IrzxrZsnlnjUB."
  },
  "overview": {
    "transformedBytes": "22",
    "transformationSummaries": [
      {
        "infoType": {
          "name": "EMAIL_ADDRESS"
        },
```

```
        "transformation": {
          "cryptoDeterministicConfig": {
            "cryptoKey": {
              "kmsWrapped": {
                "wrappedKey": "CiQAolK1/0r6aNktZPVngvs2ml/
ZxWAMXmjssvZgzSTui4keEgQSSQBaC4itVweyjdz5vdYFO3k/gh/
Kqvf7uEGYkgmVF98ZIbSffI3QRzWtR6zwLK8ZpXaDuUaQRgOuhMZJR2jf9Iq2f68aG0y
WKUk=",
                "cryptoKeyName": "projects/PROJECT_ID/locations/
global/keyRings/dlp-keyring/cryptoKeys/dlp-key"
              }
            },
            "surrogateInfoType": {
              "name": "EMAIL_ADDRESS_TOKEN"
            }
          }
        },
        "results": [
          {
            "count": "1",
            "code": "SUCCESS"
          }
        ],
        "transformedBytes": "22"
      }
    ]
  }
}
```

A couple of things to note in this output are as follows:

- `wrappedKey` is the KMS-wrapped key used to encrypt the token.
- `EMAIL_ADDRESS_TOKEN` is the encrypted token of the sensitive data (email address in our example). This is the data that you will store as well as the wrapped key, so if needed you can re-identify it later. We will see how to do that in the next step.

Step 6: Sending a re-identify request to the Cloud DLP API

This section describes how to re-identify tokenized data that you de-identified in the previous step.

To complete this task, you need the following:

- The full resource name of the Cloud KMS key that you created in *Step 1: Creating a key ring and a key*.
- The wrapped key that you created in *Step 3: Wrapping the AES key using the Cloud KMS key*.
- The token that you received in *Step 4: Sending a de-identify request to the Cloud DLP API*.

To re-identify tokenized content, follow these steps:

1. Create a JSON request file with the following text:

```
{
  "reidentifyConfig":{
    "infoTypeTransformations":{
      "transformations":[
        {
          "infoTypes":[
            {
              "name":"EMAIL_ADDRESS_TOKEN"
            }
          ],
          "primitiveTransformation":{
            "cryptoDeterministicConfig":{
              "cryptoKey":{
              "kmsWrapped": {
                "cryptoKeyName": "projects/PROJECT_ID/locations/
global/keyRings/dlp-keyring/cryptoKeys/dlp-key",
                "wrappedKey": "WRAPPED_KEY"
              }
            },
              "surrogateInfoType":{
                "name":"EMAIL_ADDRESS_TOKEN"
              }
            }
          }
        }
      ]
    }
  },
  "inspectConfig":{
```

```
    "customInfoTypes":[
      {
        "infoType":{
          "name":"EMAIL_ADDRESS_TOKEN"
        },
        "surrogateType":{

        }
      }
    ]
  },
  "item":{
    "value": "My name is Alicia Abernathy, and my email address
is TOKEN."
  }
}
```

2. Replace the following:

   - `PROJECT_ID`: The ID of your project
   - `WRAPPED_KEY`: The wrapped key that you created in *Step 3*
   - `TOKEN`: The token that you received in *Step 4*—for example, `EMAIL_ADDRESS_TOKEN(52):AVAx2eIEnIQP5jbNEr2j9wLOAd5m4kpSBR/0jjjGdAOmryzZbE/q`

   Make sure that the resulting value of `cryptoKeyName` forms the full resource name of your Cloud KMS key.

3. Save the file as `reidentify-request.json`. Now we will use this file to send a request to DLP API via `curl`.

4. Use `curl` to make a `projects.locations.content.reidentify` request:

```
curl -s \
-H "Authorization: Bearer $(gcloud auth application-default
print-access-token)" \
-H "Content-Type: application/json" \
https://dlp.googleapis.com/v2/projects/PROJECT_ID/locations/
global/content:reidentify \
-d @reidentify-request.json
```

5. Replace `PROJECT_ID` with the ID of your project.

6. To pass a filename to `curl`, you use the `-d` option (for data) and precede the filename with an @ sign. This file must be in the same directory where you execute the `curl` command.

> **Note**
> This example request explicitly targets the global location. This is the same as calling the `projects.content.reidentify` API, which defaults to the global location.

The response that you get from Cloud DLP is similar to the following JSON:

```
{
 "item": {
   "value": "My name is Alicia Abernathy, and my email address
is aabernathy@example.com."
 },
 "overview": {
   "transformedBytes": "70",
   "transformationSummaries": [
     {
       "infoType": {
         "name": "EMAIL_ADDRESS"
       },
       "transformation": {
         "cryptoDeterministicConfig": {
           "cryptoKey": {
             "kmsWrapped": {
               "wrappedKey":
"CiQAYuuIGo5DVaqdE0YLioWxEhC8LbTmq7Uy2G3qOJlZB7WXBw0SSQAjdwP
8ZusZJ3Kr8GD9W0vaFPMDksmHEo6nTDaW/
j5sSYpHa1ym2JHk+lUgkC3Zw5bXhfCNOkpXUdHGZKou1893O8BDby/82HY=",
               "cryptoKeyName": "projects/PROJECT_ID/locations/
global/keyRings/dlp-keyring/cryptoKeys/dlp-key"
             }
           },
           "surrogateInfoType": {
             "name": "EMAIL_ADDRESS_TOKEN"
           }
         }
       },
       "results": [
         {
           "count": "1",
           "code": "SUCCESS"
         }
       ],
```

```
        "transformedBytes": "70"
      }
    ]
  }
}
```

In the `item` field, `EMAIL_ADDRESS_TOKEN` from the previous step is replaced with the plain email address from the original text.

You've just de-identified and re-identified sensitive data in text content using deterministic encryption. Now let us move on and understand a few use cases of workloads where DLP can be used.

DLP use cases

You have seen how DLP can be used to inspect, identify, and re-identify sensitive data in your workloads. Now let us understand various use cases to see how DLP fits:

- **Automatically discover sensitive data**: With Cloud DLP, you can automatically understand and manage your data risk across your entire enterprise. Continuous data visibility can assist you in making more informed decisions, managing and reducing data risk, and being compliant. Data profiling is simple to set up on the Cloud console, and there are no jobs or overhead to worry about, so you can focus on the results and your business.
- **Classify data across your enterprise**: Cloud DLP can help you categorize your data, whether it's on or off the cloud, and provide the insights you need to ensure correct governance, management, and compliance. Publish summary findings to other services such as Data Catalog, Security Command Center, Cloud Monitoring, and Pub/Sub or save comprehensive findings to BigQuery for study. In the Cloud console, you may audit and monitor your data, or you can use Google Data Studio or another tool to create custom reports and dashboards.

- **Protect sensitive data as you migrate to the cloud**: You can evaluate and classify sensitive data in both structured and unstructured workloads with Cloud DLP.

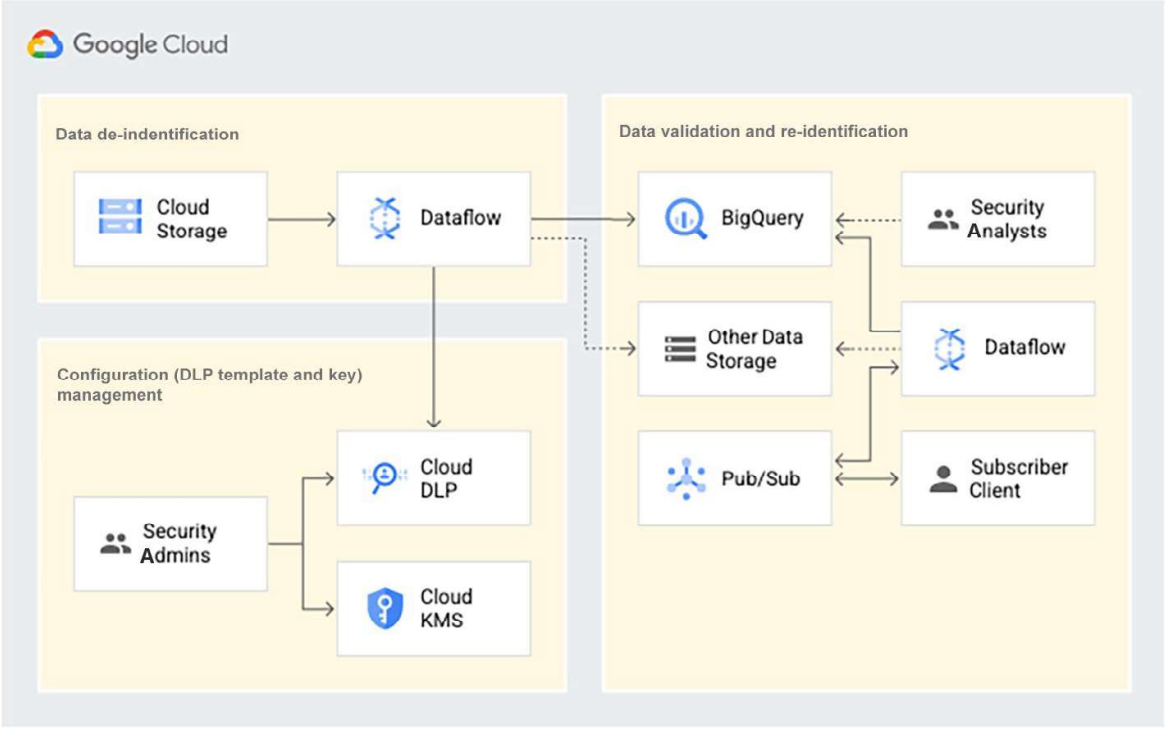

Figure 10.6 – DLP pattern for Dataflow and Data Fusion pipeline

As in *Figure 10.6*, here are a couple of patterns that you can employ to use Cloud DLP:

- Use Cloud DLP + Dataflow to tokenize data before loading it into BigQuery
- Use Cloud DLP + Cloud Data Fusion to tokenize data from Kafka Streams

By obfuscating the raw sensitive identifiers, de-identification techniques such as tokenization (pseudonymization) preserve the utility of your data for joining or analytics while decreasing the danger of handling the data. If you don't want the data to end up in the data warehouse, you can create a quarantine pipeline to send it to a sensitive dataset (with very limited permissions).

- **Use Cloud DLP in contact center chats to de-identify PII**: Cloud DLP can be used to mask any PII in customer support chats to make sure the agent doesn't get hold of any customer data.

We have seen some examples of workloads that you can use DLP for; now let us learn some best practices while using DLP.

Best practices for Cloud DLP

It can be difficult to figure out where Cloud DLP fits in your architecture or to identify requirements for Cloud DLP. Here are some best practices for you to understand how to use Cloud DLP in various scenarios:

- **Use data profiling versus inspection jobs**: Data profiling allows you to scan BigQuery tables in a scalable and automated manner without the need for orchestrating jobs. Considering the growth of data and the increasing number of tables, leveraging profiling features is recommended as it takes care of orchestration and running inspection jobs behind the scenes without any overhead. The inspection jobs can complement profilers when deeper investigation scans are needed. For example, if there are around 25,000 tables to be scanned, the recommendation is to scan all the tables with a profiler and then do a deep scan of 500 tables to flag sensitive/unstructured data that needs a more exhaustive investigation.

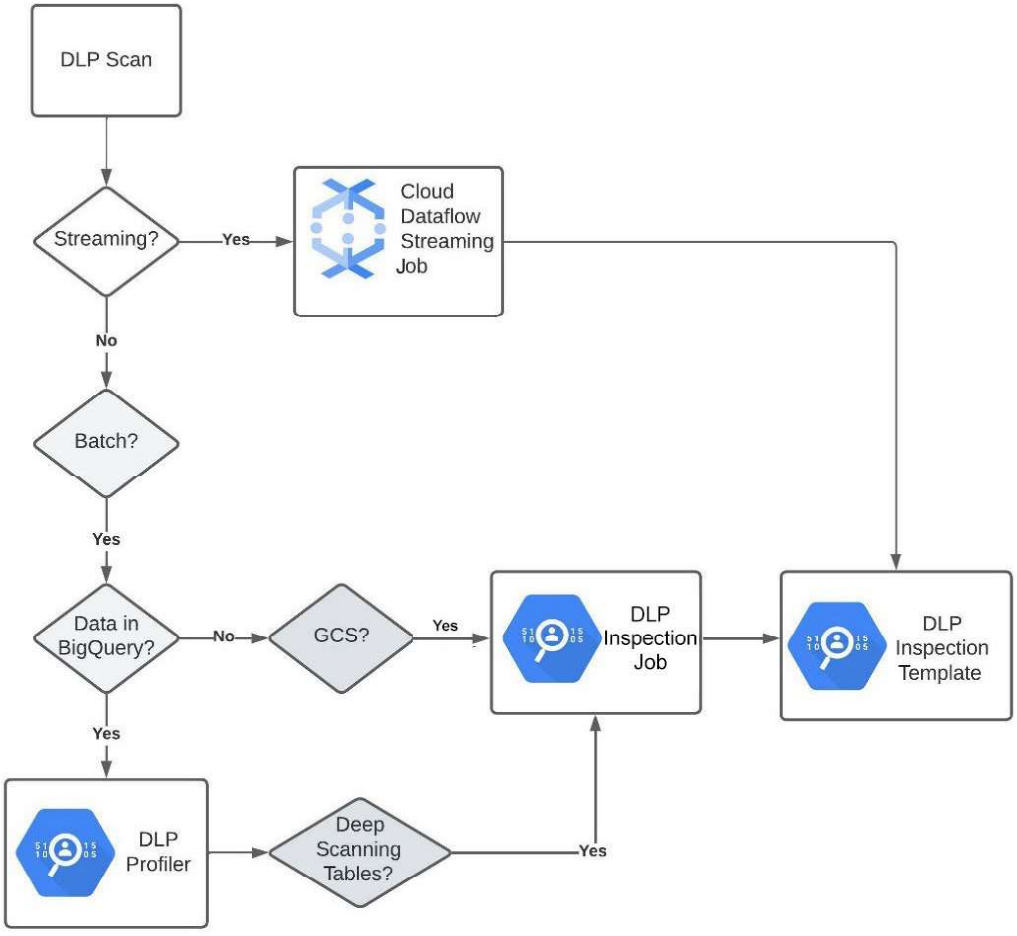

Figure 10.7 – Decision tree for inspection

Figure 10.7 shows a decision tree on when to use inspection jobs versus data profiling services.

- **Leverage built-in rules wherever possible**: Cloud DLP's built-in inspection templates should be used wherever possible. With over 150 built-in templates, Cloud DLP allows us to detect and classify data with speed and scale. For example, the `US_HEALTHCARE_NPI` tag can detect the 10-digit national provider number used for Medicare services. The use of built-in infoTypes saves the time and effort needed to create and maintain custom rules for standard infoTypes.
- **Trigger Cloud DLP scans based on events**: Cloud DLP's data profiling capability allows the automated scanning of all BigQuery tables and columns in your organization, folders, and projects. However, there might be a need to automate triggers that will enable the scan to run on new files that get uploaded to Cloud Storage or a new message that's added to Pub/Sub topics. To support this use case, Eventarc API or other BigQuery sink listeners can be configured to identify the change events that will eventually trigger a Cloud DLP scan.

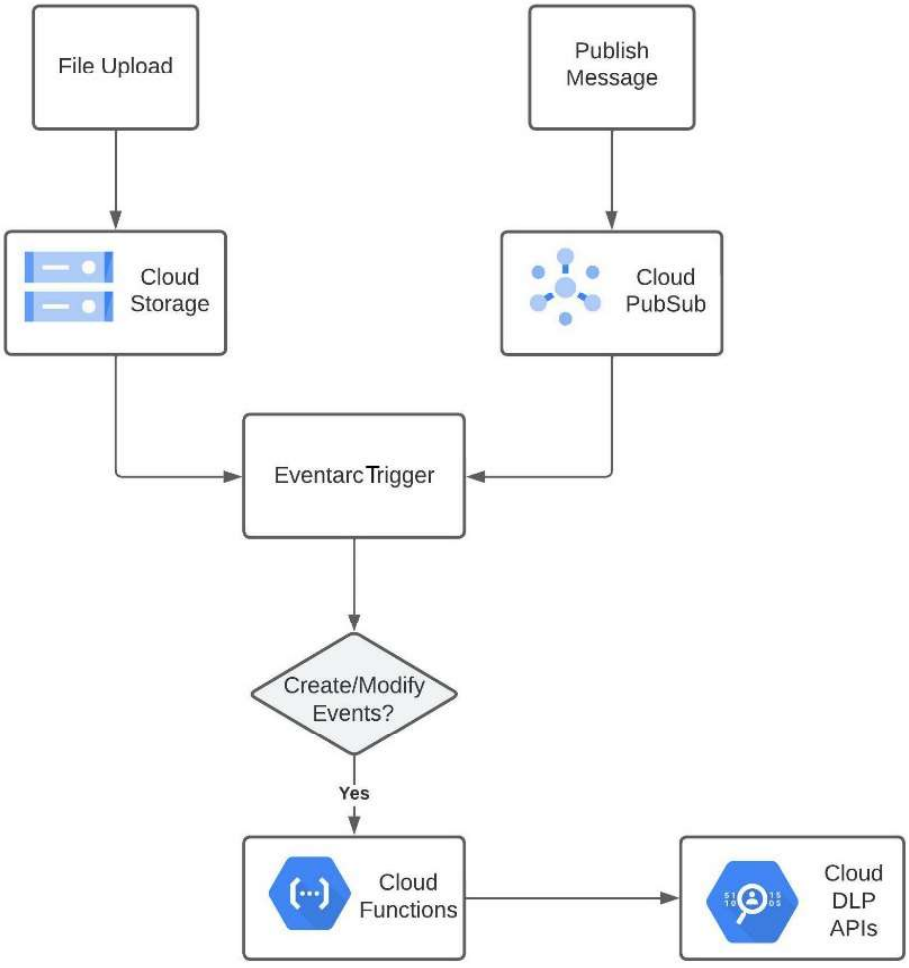

Figure 10.8 – Pattern for new updates

As shown in *Figure 10.8*, the pattern shows an example of a file upload or a Pub/Sub message triggering an Eventarc trigger. Based on the event, a cloud function will be triggered that will start the Cloud DLP scan.

- **Provide necessary permissions to the service account**: Cloud DLP creates service account agent accounts by default, which will have the necessary permissions to run Cloud DLP scan jobs. If you are scanning a highly restricted table or file, the relevant IAM roles must be added to the Cloud DLP service agent. Alternatively, if you are using a Cloud Function or Cloud Run applications to trigger Cloud DLP jobs, those service accounts must have the `dlp.jobTriggers.create` permission to successfully start the Cloud DLP scan job.
- **Publish Cloud DLP tags to Data Catalog**: Once the classification is complete, the results can be sent to Data Catalog so that the tables are tagged, only the right people get access to the data, and any sensitive data is protected. Cloud DLP allows native integration with the Data Catalog, so once we scan the BigQuery tables, it can send the tagging results into Data Catalog in the form of tag templates. The findings will be included in the summary form for the table in Data Catalog.
- **Restrict access prior to running Cloud DLP scans**: Access to any new BigQuery tables must be restricted until the Cloud DLP profiling is complete. To achieve this goal, the Cloud DLP profiler must be enabled, and only after it passes the policy checks should the data assets be moved to a state where end users can start accessing them. A quarantine state can be added to policy tags for all new tables so that users won't have access. Once Cloud DLP identifies sensitive data, a policy tag can be created in the form of a hierarchy of high, medium, and low to further restrict access depending on the content Cloud DLP identifies.

We have seen best practices for how to use Cloud DLP in various scenarios. Now let us switch gears and look at how to control data exfiltration and the best practices to prevent it.

Data exfiltration and VPC Service Controls

In the public cloud, there are several threats that organizations need to understand before deploying critical workloads. Here are a few threats that would lead to data exfiltration:

- Misconfigured IAM policies
- Malicious insiders copying data to an unauthorized destination
- Compromised code copying data to an unauthorized destination
- Access to data from unauthorized clients using a stolen credential

Here are various paths via which data can be exfiltrated in the cloud:

- Internet <-> service (stolen credentials)
  - Copy to internet
- Service <-> service (insider threat)
  - Copy from one storage service to another
- VPC <-> service (compromised VM)
  - Copy to consumer Google services
  - Copy to public GCS buckets/BigQuery dataset/GCR repo

Google Cloud offers some excellent offerings to stop the exfiltration of data as a part of its data loss prevention portfolio of products. VPC Service Controls extends a logical security perimeter around multi-tenant Google Cloud services such as **Google Cloud Storage** (**GCS**), **BigQuery**, and **Google Container Registry** (**GCR**). VPC Service Controls isolates Google Cloud-managed resources from the internet, unauthorized networks, and unauthorized Google Cloud resources. VPC Service Controls allows you to configure a security perimeter around the data resources of your Google-managed services (such as GCS, BigQuery, and Cloud BigTable) and control the movement of data across the perimeter boundary.

Let us look at the architecture of VPC Service Controls now.

Architecture of VPC Service Controls

Let us see a common VPC Service Controls architecture.

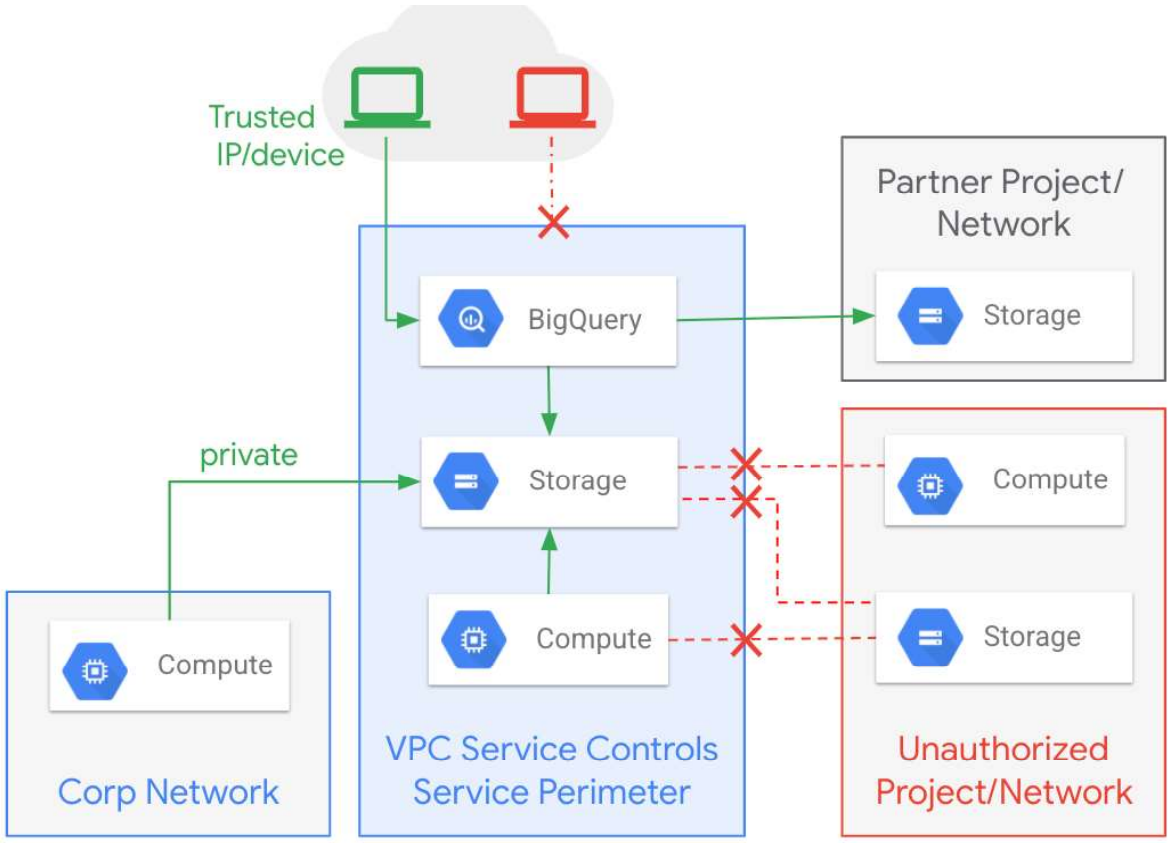

Figure 10.9 – VPC Service Controls architecture

As seen in *Figure 10.9*, the service perimeter provides a unified policy construct to isolate managed GCP resource access across the following three network interfaces:

- Internet-to-Google Cloud resource
- VPC network-to-Google Cloud resource
- Google Cloud resource-to-resource (on Google backend)

VPC Service Controls also allows you the following:

- The ability to run the configuration in dry run mode to evaluate the impact of a policy change before enforcing it.
- Fine-grained ingress and egress rules for secure data exchange between projects and other Google Cloud organizations.
- To create an access level with IP address and device attributes to enable secure access from outside the perimeter. Device attributes are typically used to establish organization trust to make sure the requests are coming from a trusted device (device attributes require BeyondCorp Enterprise).

- VPC accessible services to constrain which Google Cloud APIs (services) are accessible from a VPC network.

Now let us take a look at how VPC Service Controls can be used for API access restriction.

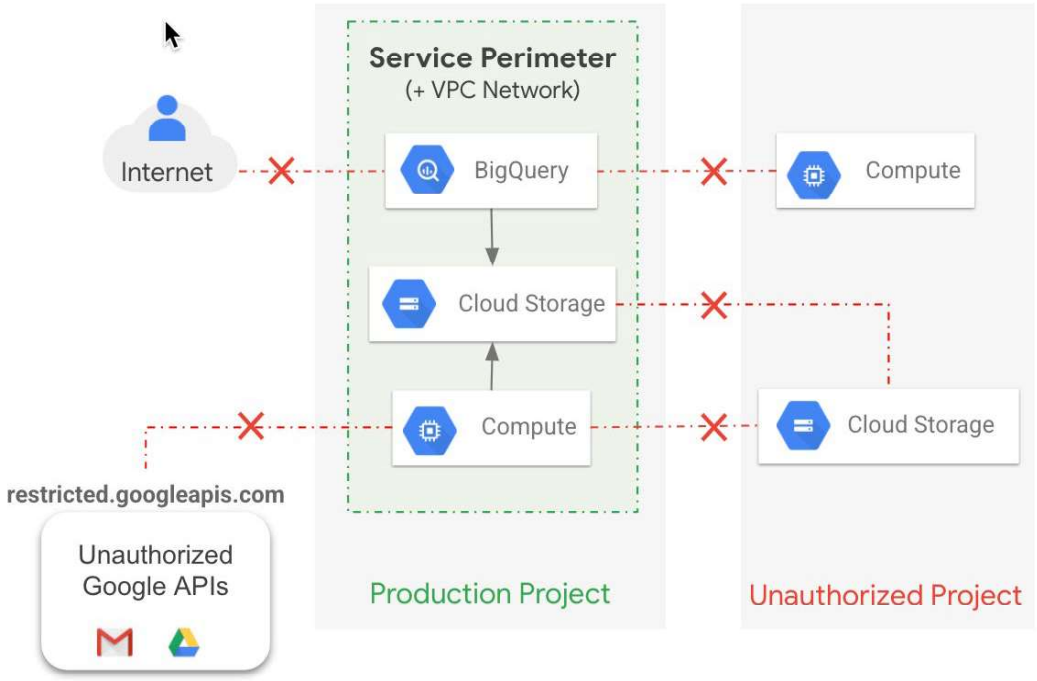

Figure 10.10 – Control APIs

As shown in *Figure 10.10*, VPC Service Controls allows you to control which APIs can be accessed from within the service perimeter:

- **Private Google Access or Private Service Connect**: Privately access Google Cloud APIs using Private Google Access.
- `restricted.googleapis.com`: Limit access to Google Cloud services that can be isolated with VPC Service Controls using the restricted **Virtual IP** (**VIP**) address range. To allow access to Cloud and Developer APIs protected by VPC Service Controls, use `restricted.googleapis.com`. The `restricted.googleapis.com` domain resolves to the `199.36.153.4/30` VIP range. The internet does not have access to this IP address range. The VIP `restricted.googleapis.com` denies access to Google APIs and services that are not supported by VPC Service Controls.
- **VPC accessible services**: Specify the exact list of APIs that can be accessed from a VPC.

Now we have seen how to restrict API access, let us see how can we allow access to the APIs within the perimeter.

Allowing access to protected resources within the VPC Service Controls perimeter

Use access levels to allow access to protected Google Cloud resources within service perimeters from outside a perimeter. An access level specifies a set of requirements that must be met in order for a request to be honored. Various variables, such as IP address and user identity, can be used to determine access levels.

To configure access levels, you'd use **Access Context Manager**. Let's take a look at it in more detail next.

Access Context Manager

Access Context Manager is a policy engine that allows administrators to create access control policies based on factors such as user geolocation and source IP address. These policies can be used as part of VPC Service Controls to ensure that a set of controls is applied to protect data in the project from adversaries.

Here are some rules when defining access levels:

- Only requests from outside a perimeter for resources of a protected service inside a perimeter are allowed.
- Access levels allow ingress only. Use an egress policy to allow access from a protected resource inside a perimeter to resources outside the boundary.
- The ingress and egress rules define which identities and resources are granted access to and from the perimeter. Use cases that formerly needed one or more perimeter bridges can be replaced and simplified with ingress and egress rules. A perimeter bridge is a feature of Google Cloud VPC Service Controls that allows users to securely and reliably connect two Google Cloud VPC networks. With this feature, users are able to create a secure connection between two VPC networks, allowing them to securely share resources, such as applications and databases, across the two networks. This provides an additional layer of security to protect resources from unauthorized access. Additionally, the perimeter bridge makes it easier to manage traffic between the two networks.
- Even though an access level normally accepts the external request, requests for a protected resource in a perimeter that originate from another perimeter are denied. You will need to add appropriate rules to get the desired action by applying ingress/egress rules.
- For IP-based allowlists, you can only utilize public IP address ranges in the access levels. Internal IP addresses are not allowed in these allowlists at the time of writing. A VPC network has internal IP addresses, and VPC networks must be referenced by its containing project via an entrance or egress rule, or a service boundary.

Now let us look at how to configure a VPC Service Controls perimeter.

Configuring a VPC Service Controls perimeter

To configure a VPC Service Controls perimeter, you should follow these high-level steps:

1. Set up access levels.
2. Create an access policy (aka perimeter).
3. Secure Google-managed resources with service perimeters.
4. Set up VPC-accessible services to add additional restrictions to how services can be used inside your perimeters (optional).
5. Set up private connectivity from a VPC network (optional).
6. Allow context-aware access from outside a service perimeter using ingress rules (optional).
7. Configure secure data exchange using ingress and egress rules (optional).

You also have the option to create an access level that will determine how your perimeter will be accessed. Let us look at that now.

Creating an access level

The following examples explain how to create an access level using different conditions:

1. Open the **Access Context Manager** page in the Cloud console and then open the **Access Context Manager** page.
2. If you are prompted, select your organization.
3. At the top of the **Access Context Manager** page, click **New**.
4. In the **New Access Level** pane, do the following:

    I. In the **Access level title** box, enter a name for the access level.

    II. Click **Add Attribute** and then select **IP Subnetworks**. The supported access level attributes are as follows:

        i. IP subnetworks

        ii. Regions

        iii. Access level dependency

        iv. Principals

        v. Device policy

    III. In the **IP Subnetworks** box, enter an IPv4 or IPv6 CIDR block—for example, **93.184.216.0/32**.

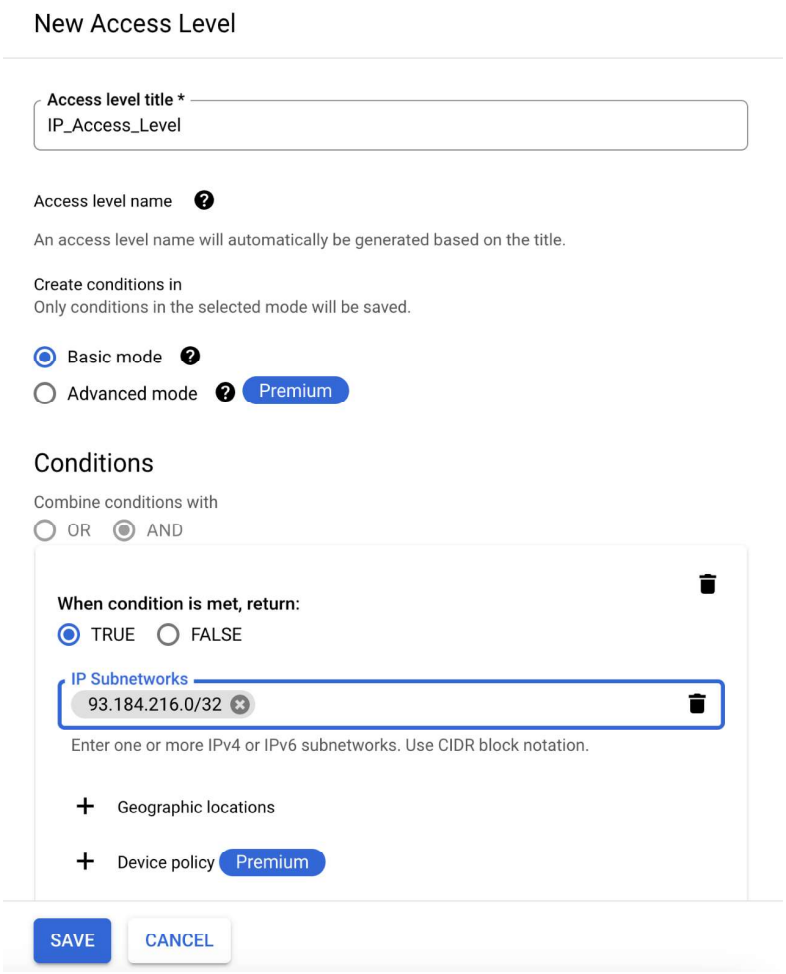

Figure 10.11 – Creating access level

Figure 10.11 shows the **New Access Level** pane.

5. Click **SAVE**.
6. The access level is saved and appears in the grid on the **Access Context Manager** page.

That's it—you just created an access level that can be used in the access policy. Now let us look at how to create an access policy.

Creating a service perimeter

Access policies are always created at an organizational level. Access policies have the following components:

- **Resources you want to include in the policy**: These are folders or projects that the perimeter will protect
- **An administrator**: This is the administrative principal (a user or a service account) that either has access to view or manage the policy

- **VPC-accessible services**: This is a service that is allowed to be accessed by other VPCs
- **Access levels**: These are, as defined in the previous sections, endpoints that will have access to the perimeter
- **Ingress policy**: A rule that allows clients outside of the service perimeter to call a protected service within the perimeter
- **Egress policy**: A rule that allows services or clients within the perimeter to call a service outside of the perimeter

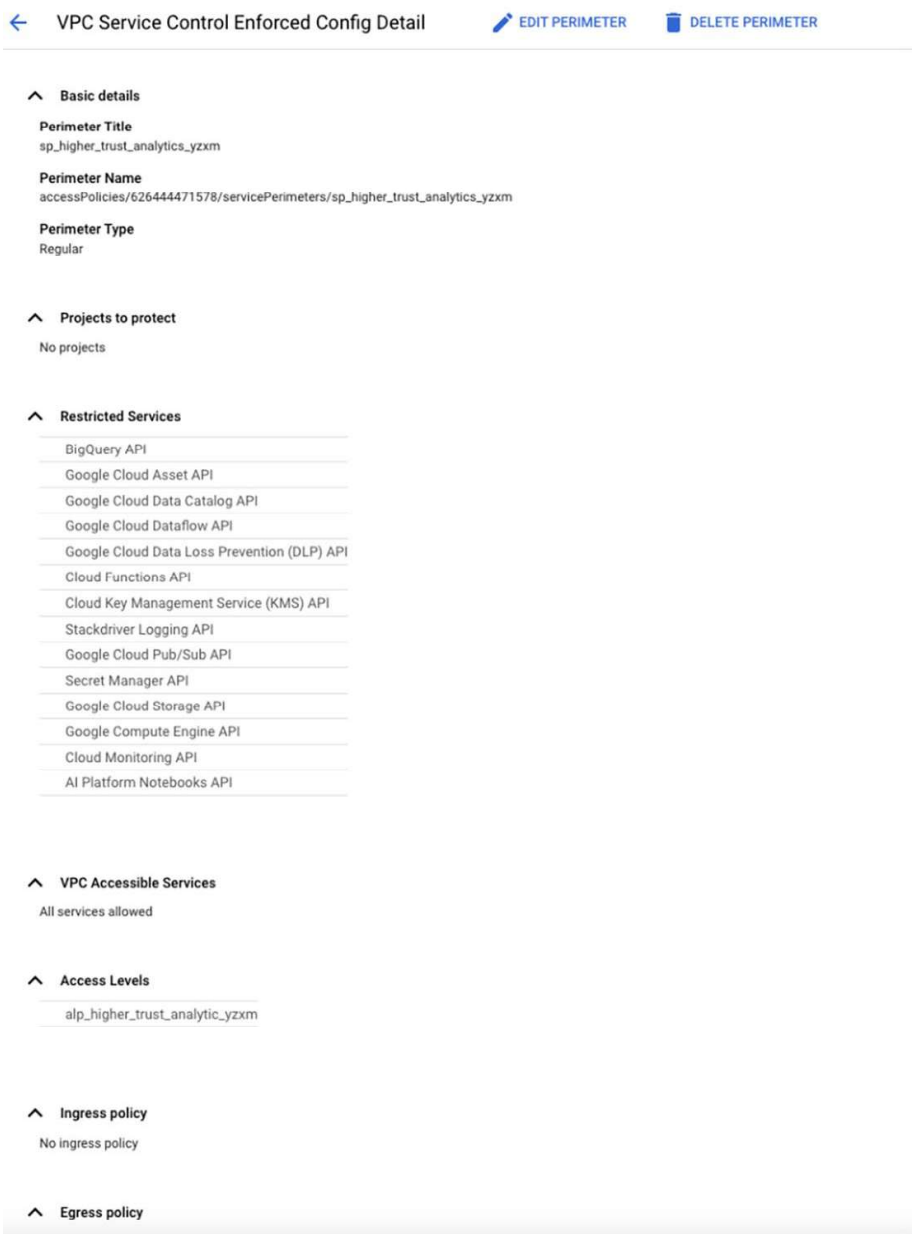

Figure 10.12 – Defining a service perimeter

As shown in *Figure 10.12*, the service perimeter will contain all the components necessary to create a perimeter.

Next, we will take a look at some of the best practices for VPC Service Controls.

Best practices for VPC Service Controls

Now that you understand the higher-level details of VPC Service Controls perimeters, let us go over some best practices:

- A single large perimeter is the simplest to implement and reduces the total number of moving parts requiring additional operational overhead, which helps to prevent complexity in your allowlist process.
- When data sharing is a primary use case for your organization, you can use more than one perimeter. If you produce and share lower-tier data such as de-identified patient health data, you can use a separate perimeter to facilitate sharing with outside entities.
- When possible, enable all protected services when you create a perimeter, which helps to reduce complexity and reduces potential exfiltration vectors. Make sure that there isn't a path to the private VIP from any of the VPCs in the perimeter. If you allow a network route to `private.googleapis.com`, you reduce the VPC Service Controls protection from insider data exfiltration. If you must allow access to a non-supported service, try to isolate the use of unsupported services in a separate project, or move the entire workload off the perimeter.
- An enterprise has many projects representing different workloads and applications. It is recommended that you organize these projects into folders.
- Allow ample time to gather data, conduct tests, and analyze violation logs. Make sure that stakeholders from your network, operations, and applications team are available for the task.
- Configure Private Google Access or Private Service Connect with `restricted.googleapis.com`.
- Move any applications using services not on `restricted.googleapis.com` to a separate project.
- Use dry run mode when configuring VPC Service Controls to an existing production environment. Make sure you understand the exceptions and errors produced by the dry run mode and see whether they match your design expectations. For example, certain service accounts such as an organization-level log sink will need to be allowed to collect logs. The dry run mode will catch these exceptions that you may not have considered in your design.
- Use the VPC Service Controls troubleshooting tool to analyze VPC Service Controls error codes.
- Set VPC-accessible services to restricted services to prevent access to unrestricted services.

- Ideally, do not include on-premises corp networks with user devices within a very secure service perimeter configuration.
- Document the following for every use case you use VPC Service Controls for:
  - The access pattern
  - The actors that can trigger the use case
  - Conditions that trigger the use case
  - Whether the use case is a valid access pattern and should be allowed
  - Any assumptions that pertain to the use case
  - Treat VPC Service Controls logically the same way you would a firewall; in the network, you need to "know your flows" to successfully secure the environment with the required ingress and egress flows enabled

VPC Service Controls is an important part of your cloud security strategy, so make sure you spend enough time understanding it.

Summary

We have covered a lot of things in this chapter. We went over various DLP definitions, use cases, and architecture options. We went over in detail how to use DLP for inspection and de-identification. We also saw examples of how to call DLP APIs and how to interpret the response. Last but not least, we went over data exfiltration strategies such as VPC Service Controls and their best practices. In the next chapter, we will go over the features of Secret Manager and how it can be used to store your application secrets securely.

Further reading

For more information on Cloud DLP and VPC Service Controls, refer to the following links:

- InfoType detector reference: `https://packt.link/cVC5r`
- Access level design: `https://packt.link/ZAEk4`
- Creating device-based access levels for VPC Service Controls: `https://packt.link/MpHwV`
- Ingress and egress rules for VPC Service Controls: `https://packt.link/3bBkP`