3. Click **Create**.

   Figure 8.12 shows how your configuration should look once you have populated the firewall rule settings as per the preceding example. You only specify `22` or `3380` if required under **Protocols and ports**. The example illustrates how you can open both SSH and RDP ports.

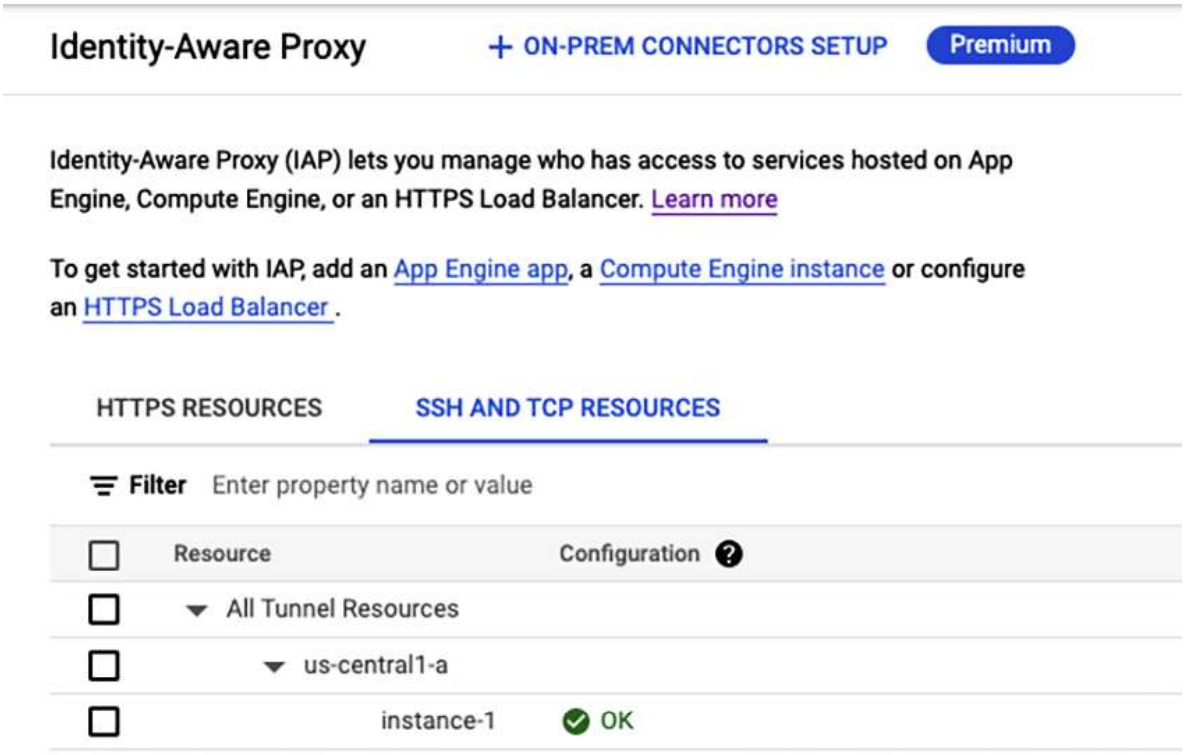

Figure 8.12 – SSH and TCP resources after creating the firewall rule successfully

In *Figure 8.12*, observe that after applying the firewall rule, when we navigate back to **SSH AND TPC RESOURCES** under the IAP settings, an `OK` message is presented. The `Error` message is no longer displayed.

Next, we need to give access permissions to the users. You can offer access to all VMs in a project – to a user or a group – by establishing IAM permissions at the project level. Let's take a look at what's involved:

4. Navigate to the IAP admin page from the left menu under the **Security** sub-menu.
5. From there, you can select the **SSH and TCP Resources** tab.
6. Then, you need to select the VM instances that you want to configure. You can click on **Show info panel** if the info panel is not visible. This panel comes up on the right-hand side of the console.

7. We will next look at how you can add a member and configure them. Refer to *Figure 8.13* on how to add a new member and apply the relevant role.
8. We add a new member that can either be an email address or a group email address that you have the option to configure in Google Cloud Identity.
9. Once you have added the user or group, you can set **Role** to `IAP-secured Tunnel User`.

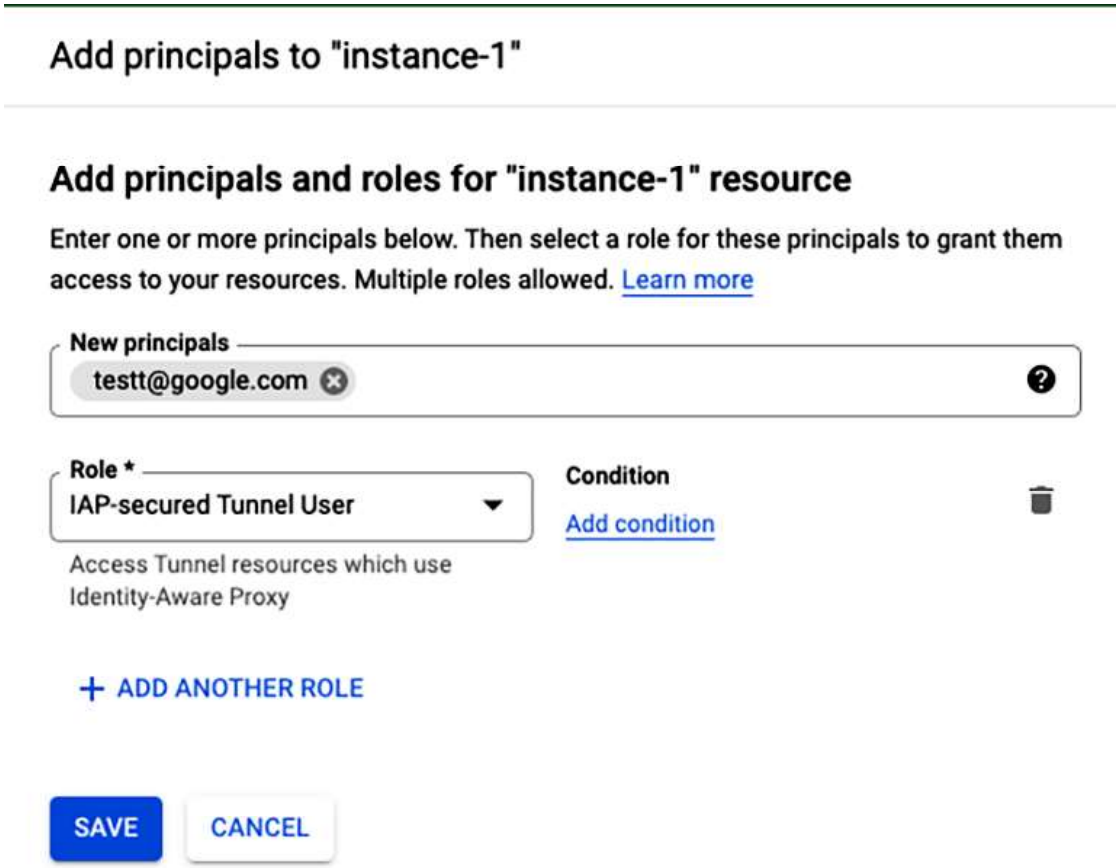

Figure 8.13 – Adding a principal and applying the IAP-secured Tunnel User role

You can also optionally create more fine-grained permissions by using the **Add condition** feature and configure a member restriction.

> **Note**
>
> You can refer to *Chapter 6*, *Google Cloud Identity and Access Management*, to learn more about how to add conditions to an IAM principal.

10. Once finished, you can click **SAVE**.

> **Note**
>
> In the preceding step, *Step 9*, we are referring to the access levels that can be created using Access Context Manager. Within Access Context Manager (which is a different product), you have the option of configuring policies based on access levels. For example, you can create an access policy for all privileged users called `PrivAccess`. Here you can specify certain parameters – you might want to restrict access to users only coming from a certain IP range or geo-location, for example. With the advanced features that you can get as part of Access Context Manager using the Beyond Corp Enterprise solution, you can also configure a policy that can do endpoint checks, such as checking for X.509 certifications, ensuring the operating system is approved and patched, hard disk encryption is enforced, and so on. This is part of Access Context Manager and is outside the scope of the Google Cloud Professional Security Engineer exam.

TCP forwarding requires different permissions based on how the user plans to utilize it. On the other hand, the user would need the following rights in order to connect to an unauthenticated VM using `gcloud compute ssh`:

- `iap.tunnelInstances.accessViaIAP`
- `compute.instances.get`
- `compute.instances.list`
- `compute.projects.get`
- `compute.instances.setMetadata`
- `compute.projects.setCommonInstanceMetadata`
- `compute.globalOperations.get`
- `iam.serviceAccounts.actAs`

If you don't have an external IP address, you can connect to Linux instances through IAP. The `gcloud compute ssh` command can be used to establish a secure connection to your instance. For TCP tunneling to work, your instance's access settings (specified by IAM permissions) must permit it:

```
gcloud compute ssh instance-1
```

IAP TCP tunneling is used automatically if an external IP address is not provided by the instance. If the instance has an external IP address, IAP TCP tunneling is bypassed in favor of the external IP address. You can use the `--tunnel-through-iap` flag so that `gcloud compute ssh` always uses IAP TCP tunneling.

RDP traffic can be tunneled using IAP to connect to Windows instances that do not have an external IP address.

In order to connect to a VM's remote desktop, you can use IAP TCP forwarding with IAP Desktop:

1. In the application, select **File | Add Google Cloud project**.
2. Enter the ID or name of your project and click **OK**.
3. In the **Project Explorer** window, right-click the VM instance you want to connect to and select **Connect**.

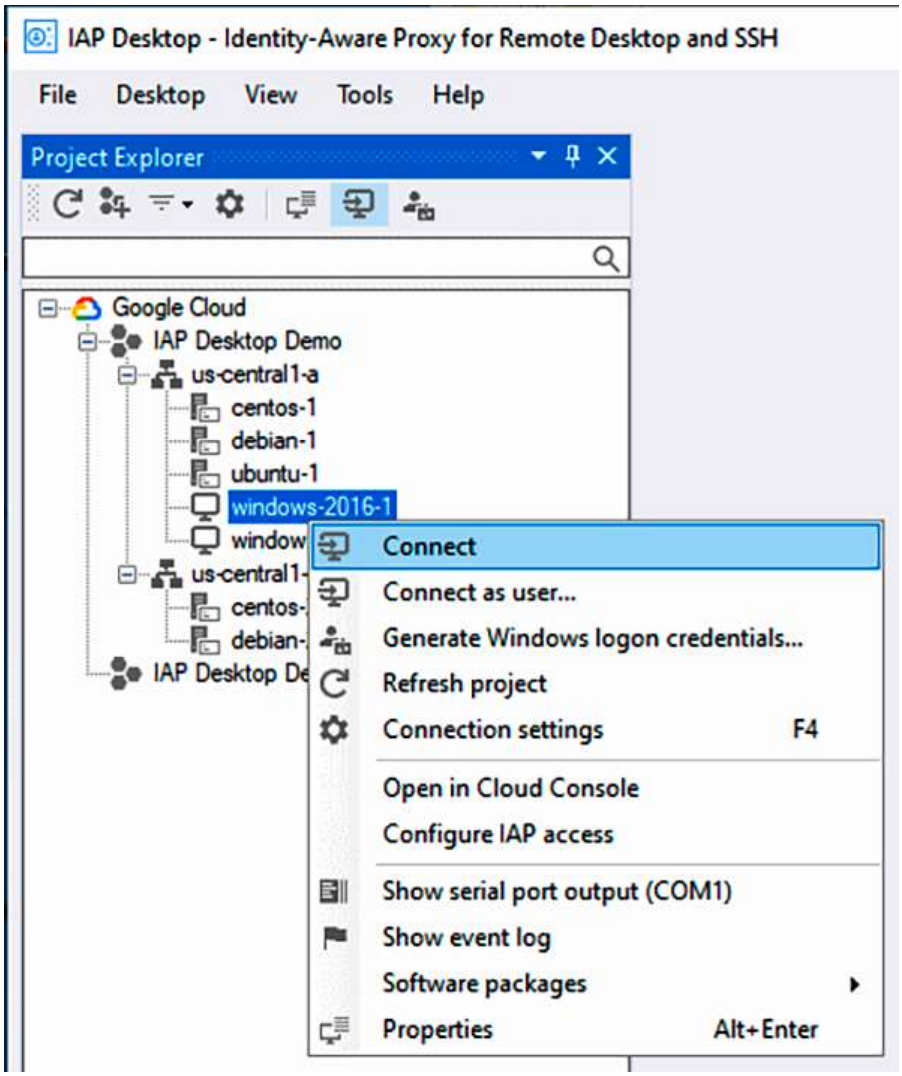

Figure 8.14 – Configuring IAP Desktop for IAP

Figure 8.14 is a screenshot of the IAP Desktop tool, which can be used to establish RDP connections for your Windows instances.

> **Note**
>
> You can find more information about IAP Desktop from its GitHub page at `https://packt.link/lZh5q`.

We will next move on to the topic of Cloud NAT. Here we will look at how NAT is used with some examples and gain a better understanding of the topic.

Cloud NAT

Cloud NAT is a topic that does not appear a lot in the exam. However, it is important to know how it works and the use cases for why you would need NAT. We will also look at the Google Cloud implementation of NAT architecture, which is different from the traditional NAT architecture.

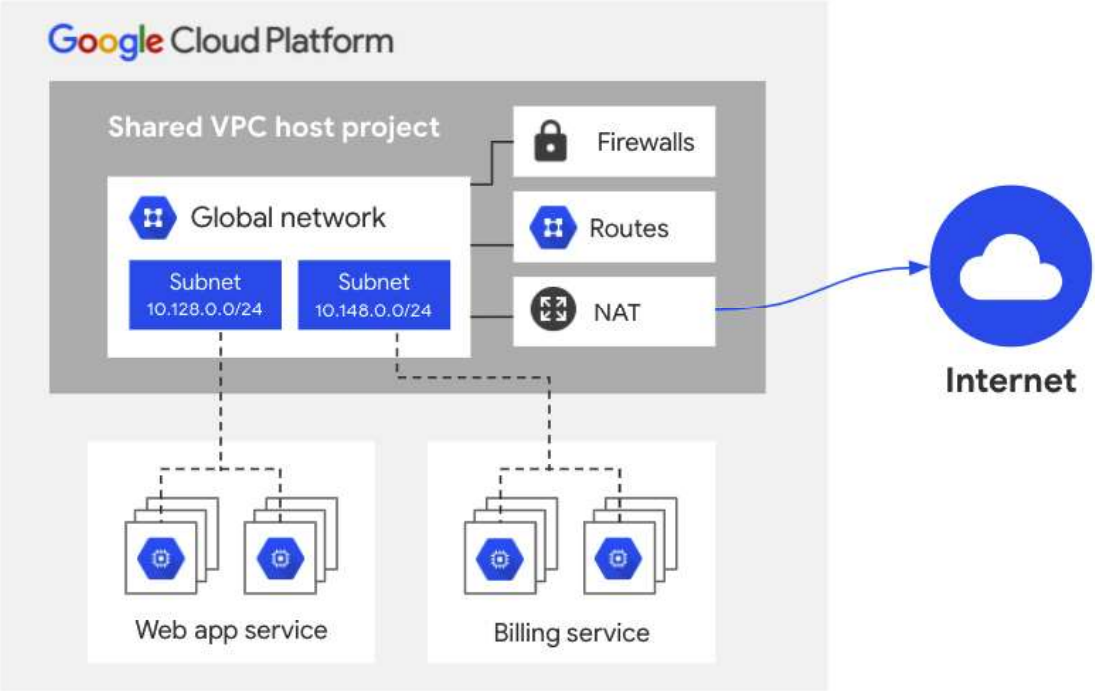

Figure 8.15 – Cloud NAT allowing outbound connections only to the internet

Figure 8.15 shows how Google Cloud NAT works. Cloud NAT is offered as a managed service that provides high availability and seamless scalability. It allows outbound connections only to the internet, whereas inbound traffic is allowed only if it is in response to a connection initiated by an instance. Cloud NAT is a regional resource, fully distributed and software-defined. There are no intermediate NAT proxies in the data path. NAT configuration is stored in the control plane and is pushed to the hosts; this means NAT keeps working regardless of the control plane state, and there are no choke points or performance penalties. Cloud NAT also supports alias IPs on VMs and relies on the default internet route in VPC.

It is important to note that there are cases when NAT is not performed even when configured. They are as follows:

- The VM has an external IP.
- The default internet route has changed.
- There's communication between backend VMs and the load balancer proxy.
- There's communication with Google APIs (Private Google Access is used instead).

Next, we will look at how you can create a Cloud NAT instance for a GCE instance that does not have an external IP address and wants to communicate with the internet. There are eight key steps involved in the configuration:

1. First, create a VPC network and a subnet as covered in *Chapter 7*, *Virtual Private Cloud*.
2. Then, create a VM instance without an external IP address. It is important to select the **No External Interface** option under the **Networking** tab of the GCE instance creation wizard.
3. Create an ingress firewall rule to allow SSH connections. This is similar to how we created a firewall rule to allow IAP in the IAP section. Ensure that the source IP range for IAP is specified, that is, the range `35.235.240.0/20`.
4. Navigate to the **Identity-Aware Proxy** settings and under **SSH & TPC RESOURCES**, select **Add Member** and give the `IAP-secured Tunnel User` permission. Again, this step is similar to what we configured in the *Using Cloud IAP for TCP forwarding* section.
5. In this step, we will navigate to the GCE page and log in via SSH from the console to the GCE instance that we created in *Step 2*. Once you have successfully logged in to the GCE instance via SSH, from the **command-line interface** (**CLI**), issue the `curl example.com` command. This should not result in any output.

6. Next, we will create a NAT configuration using Cloud Router. The NAT configuration is very straightforward. Once you have logged in to your Google Cloud console and the project is selected, you can go to the **Cloud NAT** page (**Networking | Network Services | Cloud NAT**). Once there, click on **Get started** or **Create NAT gateway**. You may see a welcome message if you are accessing this page for the first time. If you have already visited the page in the past, you will not see the welcome message; do not be confused if you don't see the following message. You should see a screen similar to the one shown in *Figure 8.16*.

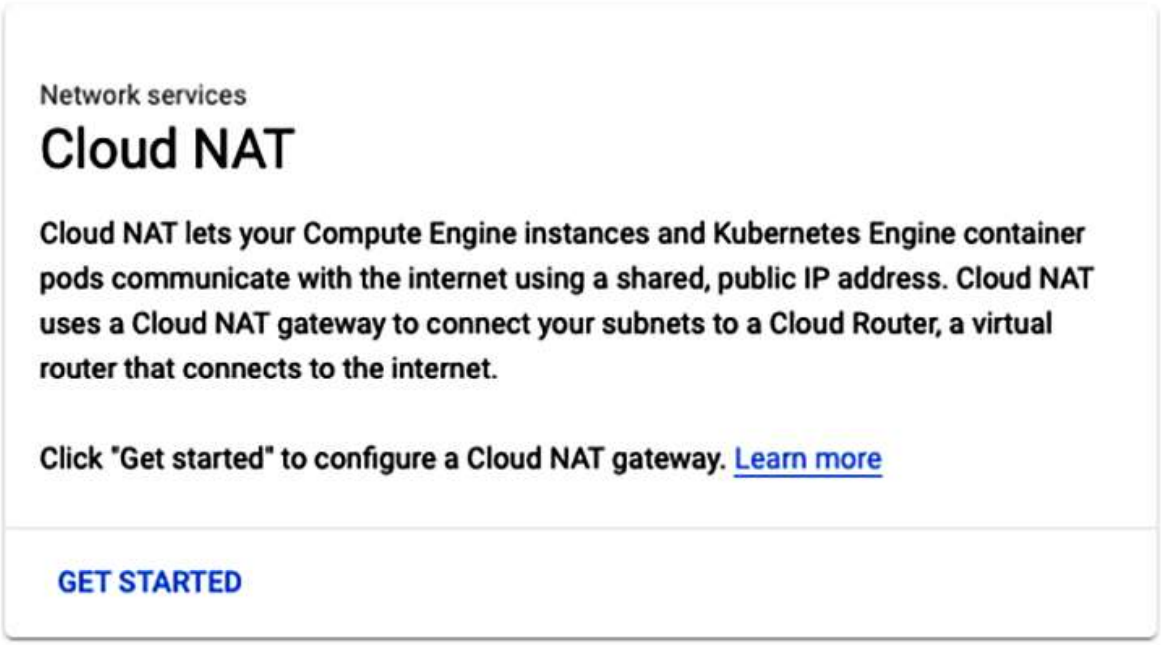

Figure 8.16 – Getting started with Cloud NAT

7. Refer to *Figure 8.17*, where you see the various properties you can configure when you create a Cloud NAT gateway. Here enter the name for your gateway; for this example, we are using the name `nat-config`. Then, specify your network; for this example, we have set **VPC network** to `startshipnw`. In your configuration, it will be the network that you have created, or you can even use the default network. Next, set **Region** to `us-east1 (your region)`. For **Cloud Router**, select `Create new router`. Then, enter the name of your preference; in our example, we have used the name `my-router`. Once finished, click **CREATE**.

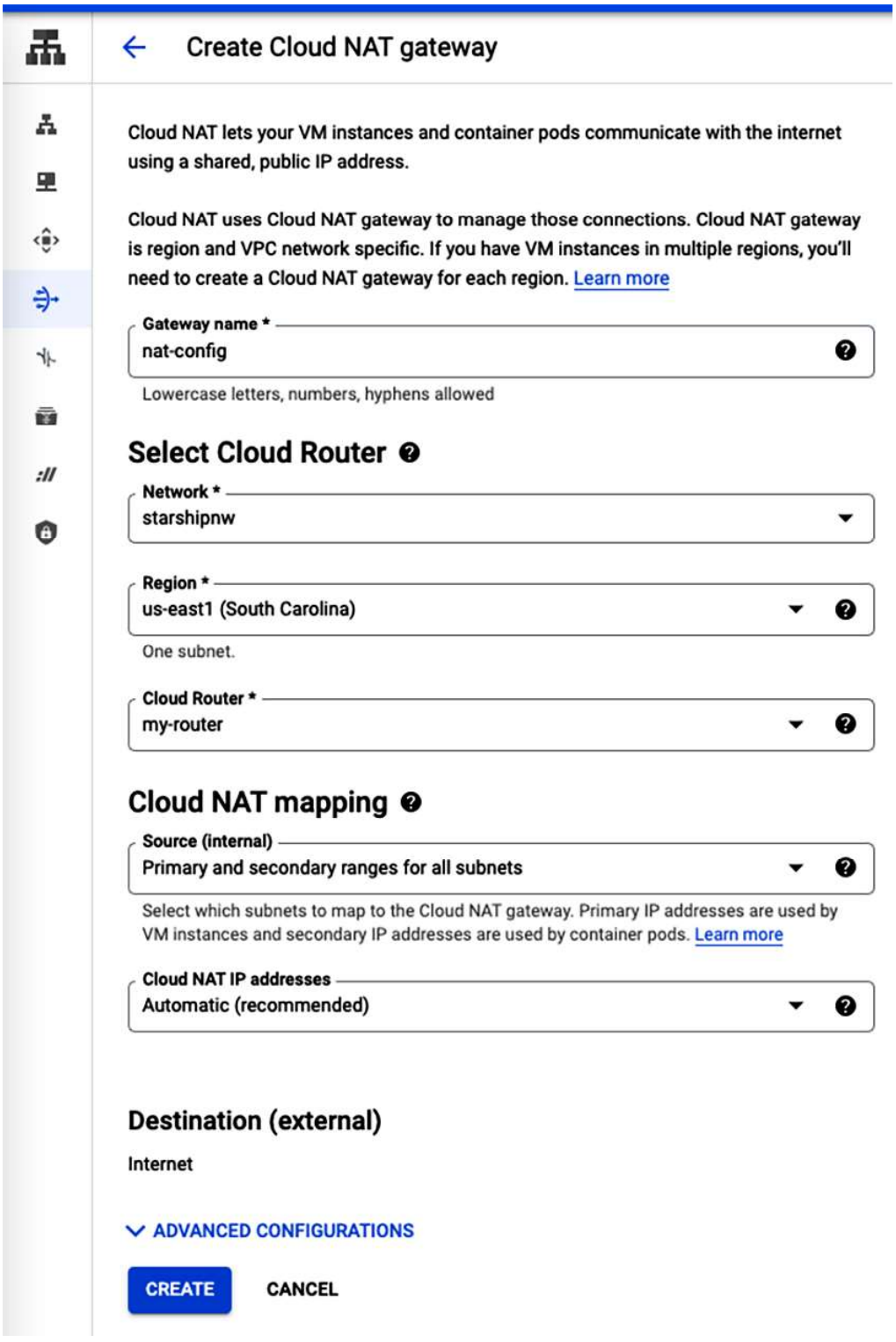

Figure 8.17 – Configuration to create a Cloud NAT gateway

8. Once the steps are completed, we can now go back to our instance under GCE, repeat the test, and issue the `curl example.com` command. This time, we will see an output similar to the following:

```
<html>


<head>

<title>Example Domain</title>

…

…

…

</head>

<body>

<div>

<h1>Example Domain</h1>
<p>This domain is established to be used for illustrative
examples in documents. You can use this domain in examples
without prior coordination or asking for permission.</p>

<p><a href="http://www.iana.org/domains/example">More
information...</a></p></div>

</body>

</html>
```

This concludes the process to create a Cloud NAT instance, which will now allow your VM that was configured with no internet access to access the internet. We will now move on to learn about Cloud Armor, which provides DDoS mitigation and a WAF function.

Google Cloud Armor

This section covers DDoS protection and the use of WAFs to provide safety for your web-based infrastructure. You can protect your Google Cloud workloads from a wide range of threats, including DDoS attacks and application attacks, such as XSS and SQL injection, with Cloud Armor (SQLi). Some capabilities are built in to provide automated protection, while others require manual configuration. We will look at those capabilities of WAFs in more detail in this section:

Figure 8.18 – How Google Cloud Armor secures your infrastructure

Cloud Armor leverages Google's global and distributed infrastructure to detect and absorb attacks and filter traffic through configurable security policies at the edge. It should be kept in mind that several aspects of Google Cloud Armor are only available for applications running behind an external HTTP(S) load balancer. *Figure 8.18* illustrates the placement of Cloud Armor, which is in line with the external load balancer, providing the ability to create IP allow and deny lists, enforce geo-location-based rules, and configure the WAF and custom rules from Layer 3 to Layer 7.

There are some Layer 3 and Layer 4 protections built into Cloud Armor that do not require any user configuration. These pre-built rules provide protection against volumetric attacks such as DNS amplification, SYN floods, Slowloris, and so on.

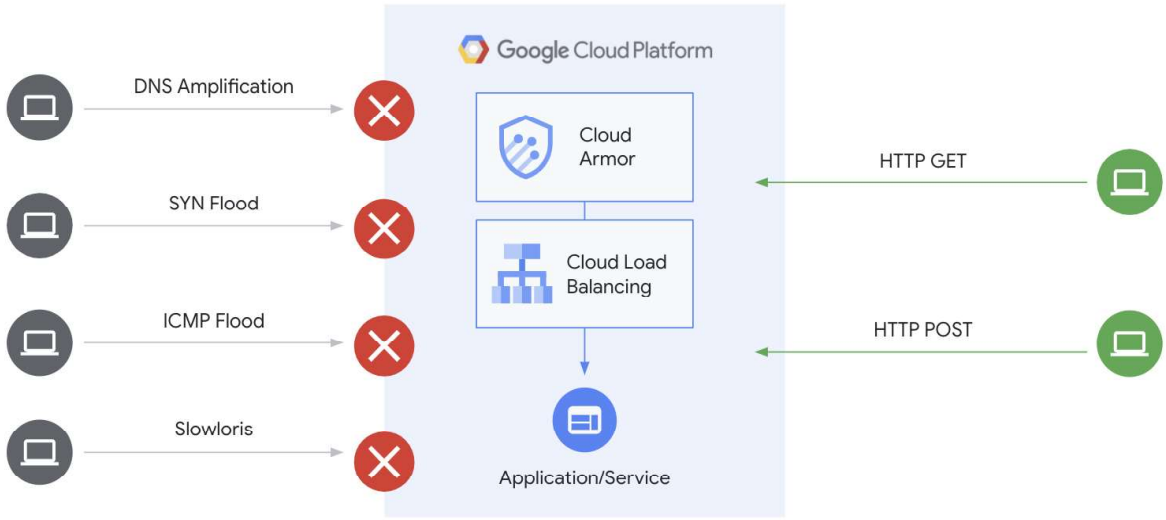

Figure 8.19 – DDoS protection against volumetric attacks

Figure 8.19 illustrates the built-in security policies for Cloud Armor that provide protection against these common types of volumetric attacks.

Next, we will look at how Cloud Armor works and some deployment models that will help you understand the use cases. As mentioned earlier, Google Cloud Armor is always in line and scales to match Google's global network. It protects applications and services behind external HTTP(S) load balancers, SSL proxy load balancers, or TCP proxy load balancers from volumetric DDoS attacks on the network or protocol. Because it can detect and counteract network threats, Cloud Armor's load balancing proxies will only allow properly formatted requests. Custom Layer 7 filtering policies, including preconfigured WAF rules to limit OWASP Top 10 web application vulnerability risks, are also available to backend services behind an external HTTP(S) load balancer. Google Cloud Edge can be used to define rules that enable or restrict access to your external HTTP(S) load balancer, as near as is feasible to the source of incoming traffic, through the use of security policies. A firewall can be set up to block unwanted traffic from accessing or consuming resources in your private cloud.

Cloud Armor is available in two tiers, called **Managed Protection Standard** and **Managed Protection Plus**. The key difference between them is the DDoS response team and cost protection that Plus provides compared to the Standard edition. There is no difference in the capabilities of the product.

Let's take a look at the Cloud Armor deployment models before we deep dive into some technical aspects such as security policies and WAF rules.

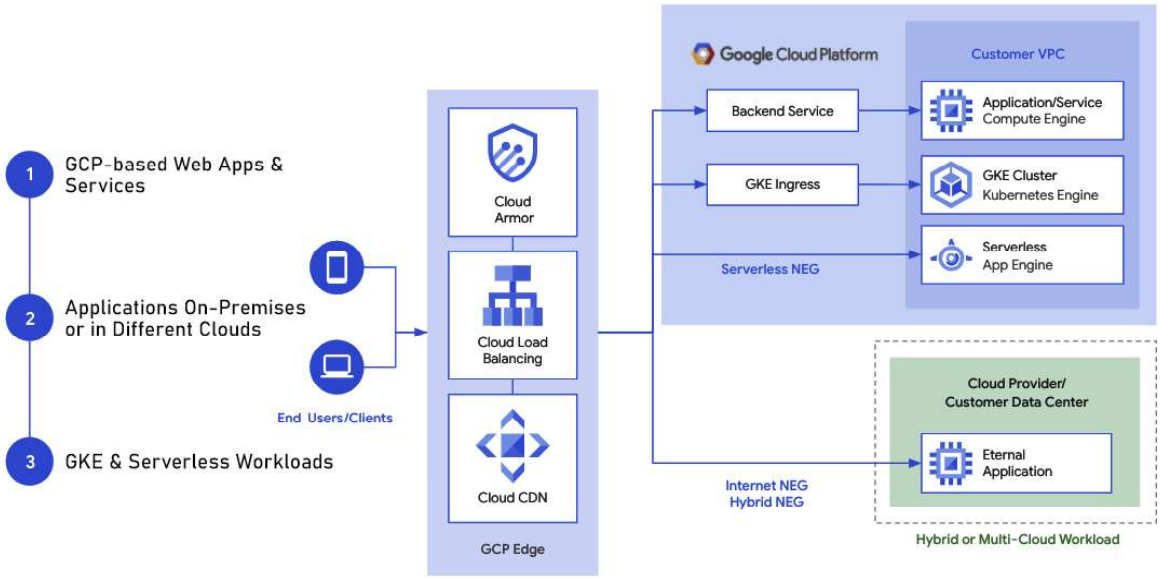

Figure 8.20 – Cloud Armor deployment models

We will use *Figure 8.20* to look at the deployment models:

- Cloud Armor supports the protection of Cloud **Content Delivery Network** (**CDN**) origin servers by enforcing Cloud Armor security policies on dynamic requests as well as cache misses destined for the CDN origin server. Most applications are often complex, serving both cacheable static content as well as dynamic requests from the same services. Cloud Armor is a security service that can help protect websites and applications from malicious attacks. It is able to inspect and filter requests that are sent to the origin servers of CDN. This helps to prevent so-called *cache-busting* attacks, which are attempts to bypass the cache and force the origin servers to send updated content. Additionally, Cloud Armor can help protect dynamic portions of websites and applications from the 10 most common security vulnerabilities as defined by OWASP.
- There is often a need to enforce a consistent set of security controls to applications no matter where they are deployed, whether they are migrating to Google Cloud or running in a hybrid configuration. With internet **Network Endpoint Groups** (**NEGs**), you can leverage all of Google's edge infrastructure, including global load balancers, Cloud CDN, and Cloud Armor, to protect your website or applications no matter where they are hosted. Cloud Armor can help protect your applications, whether from DdoS attacks or other common web attacks, without you having to deploy your applications on Google Cloud.

> **Note**
>
> NEGs do not fall within the scope of the exam. You can refer to the *Further reading* section to read more about NEGs and how to configure them.

- With GKE Ingress support for Cloud Armor, you can help protect your containerized workloads by placing them behind Google's global load balancers and configuring Cloud Armor security policies for Layer 7 filtering and WAF use cases.

We will now look at the key components of Cloud Armor in more detail, with examples of how to configure them.

Security policies

In Cloud Armor, you can build security policies that can help you defend applications operating behind a load balancer against DDoS and other web-based assaults, regardless of whether the apps are deployed on Google Cloud or in a hybrid or multi-cloud architecture. It is possible to stop traffic before it reaches your load-balanced backend services or backend buckets using security policies that filter Layer 3 and Layer 7 requests for common web exploits. An IP address, an IP range, a region code, and request headers are only a few examples of the conditions used to filter traffic by security policies.

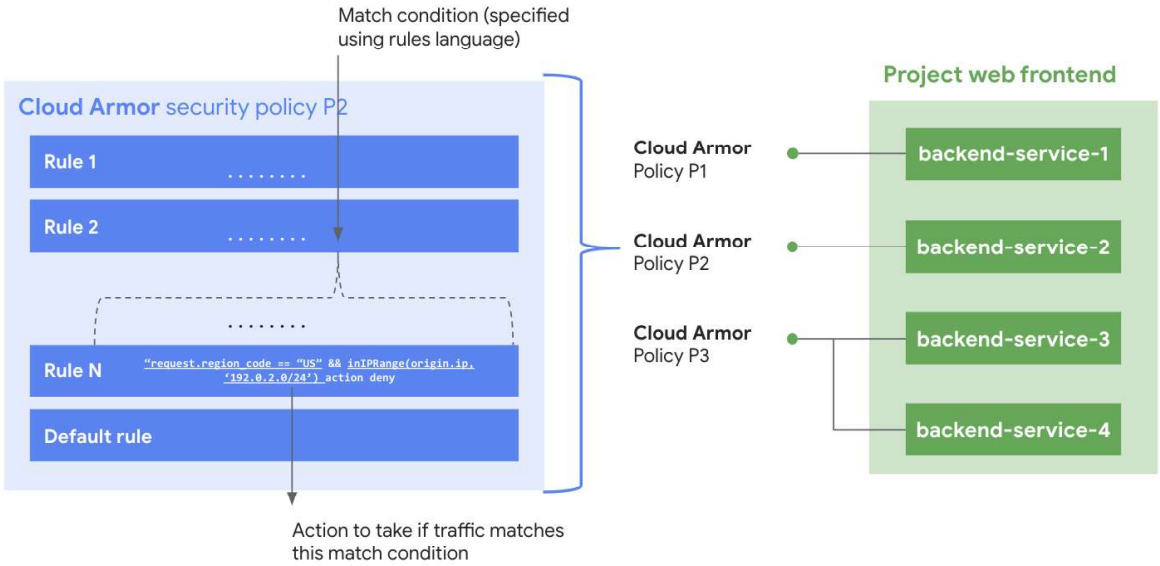

Figure 8.21 – Cloud Armor security policies

This section will take a closer look at security policies, looking at factors such as the requirements before you develop a policy and some of security policies' most important features (as depicted in *Figure 8.21*). Backend services behind an external load balancer are only able to access the security policies available to those services. Instance groups, serverless NEGs, internet NEGs for external services, and Google Cloud Storage buckets are all supported backend services for Google App Engine, Cloud Run, and Cloud Functions. You must employ internet NEGs for all of your hybrid or multi-cloud protection needs.

Security policies cannot be applied until the load balancer is an external HTTP(S) load balancer and the backend service's external load balancing scheme and the backend service's protocol are met – that is, the protocol must be either HTTP, HTTPS, or HTTP/2.

There are two categories of security policies that you can configure **– backend security policies** and **edge security policies**. Instance groups or NEGs, including internet, zonal, and serverless NEGs, can be used to filter and protect backend services. The actions associated with backend security policies are allow, deny, redirect (to Google reCAPTCHA), throttle, and rate-based ban for each source IP range or geography. You can also create WAF rules and named IP lists, including Layer 7 filtering and adaptive protection (adaptive protection is a new feature and currently out of scope for the exam).

For edge security policies, you can create and allow or deny based on source IP and source geography. These policies allow users to establish access control and filtering policies for cached information. When compared to backend security rules, edge security policies only allow for filtering based on a handful of criteria. A backend policy cannot be used to set an edge security policy.

At the edge of Google's network – upstream of the Cloud CDN cache – edge security measures are installed and implemented. Two levels of security can be provided by coexisting backend security policies and edge security policies. Regardless of the resources that a backend service refers to, they can all be applied simultaneously (for example, instance groups or network endpoint groups). In order to secure backend buckets, you must use edge security policies. You should keep in mind that edge security policies are examined and executed prior to IAP. An edge security policy blocks a request before the identity of the requestor is evaluated by IAP.

Rule evaluation order for a security policy is determined by rule priority. When determining which rules are examined first, look at the rule with the lowest numeric value. This rule has the highest logical priority and is evaluated first.

If none of the higher priority rules is met, or if no other rules are present, a default rule is applied to all security policies. A priority of `2147483647` is automatically assigned to the default rule, which is always included in the policy.

Although the default rule cannot be deleted, you can change the default action associated with the rule. It is possible to change the default rule's action from allow to deny.

Use the Google Cloud Armor custom rules language to define expressions in the match condition of one or more rules. When Cloud Armor receives a request, it analyzes it against the following expressions. Depending on whether the rules are met, inbound traffic is either denied or allowed. The following are some instances of Google Cloud Armor expressions written in the **Common Expression Language** (**CEL**):

- To define expressions in a rule, use the `gcloud --expression` flag or the Google Cloud console.
- In the following example, requests from `2001:db8::/32` in the `AU` region match the following expression:

```
origin.region_code == "AU" && inIpRange(origin.ip,
'2001:db8::/32')
```

- The following example matches with requests from `192.0.2.0/24` and with a user agent that contains the string `WordPress`:

```
inIpRange(origin.ip, '192.0.2.0/24') && has(request.
headers['user-agent']) && request.headers['user-agent'].
contains('WordPress')
```

WAF rules

In this section, we will look at the different types of rules that you can create. There are some WAF rules that come pre-configured, but you do have the option of creating custom rules to meet your requirements. Let's take a look at the options available.

You can create IP-address-based rules both at the edge and for your backend services. These rules are the IP address `allowlist-` and `denylist`-based rules. Both Ipv4 and Ipv6 are supported. HTTP `403` (Unauthorized), `404` (Access Denied), or `502` (Bad Gateway) responses are all possible outcomes of deny rules. An HTTP `429` error is an error code that is returned when an application has exceeded a set number of requests. This is usually caused by an action rule that limits the number of requests that can be made in a given period of time.

There are pre-configured rules based on the OWASP ModSecurity core rule set version 3.0. These rules can provide you with protection from the following types of attack:

- SQL injection
- **Cross-site scripting** (**XSS**) attacks
- **Local file inclusion** (**LFI**) attacks
- **Remote file inclusion** (**RFI**) attacks
- **Remote code execution** (**RCE**) attacks

Use rate-limiting rules to restrict requests per client based on the threshold you define, or temporarily ban clients who exceed the request threshold for the duration you set.

There are bot management rules as well (in preview only), which are out of the scope of the exam. Pre-configured rules also exist for named IP lists; we will cover them in more detail in the *Named IP lists* section.

> **Note**
>
> You can fine-tune the pre-configured WAF rules or write custom rules. For the purpose of the exam, you need to know what types of rules exist and how they work. You are not tested on writing custom WAF rules.

Named IP lists

Using the named IP list feature in Cloud Armor, you can reference third-party maintained IP ranges and IP addresses within a security policy. You don't have to specify individual IP addresses. Instead, you can reference an entire list containing multiple IP addresses that you would like to allow or deny. It is important to note that named IP lists are not security policies, but they can be specified in a policy.

It's possible to build security rules that allow all IP addresses that are in the provider list and reject all IP addresses that are not in the provider list: `ip1`, `ip2`, `ip3`....`ipN`:

```
gcloud beta compute security-policies rules create 1000 \
    --security-policy POLICY_NAME \
    --expression "evaluatePreconfiguredExpr('provider-a')" \
    --action "allow"
```

Custom named IP address lists are not possible to construct. Only third-party providers who work with Google and maintain named IP address lists can use this capability. CloudFlare and Imperva are a couple of the providers who partner with Google Cloud to supply named IP lists for Google's named IP list service.

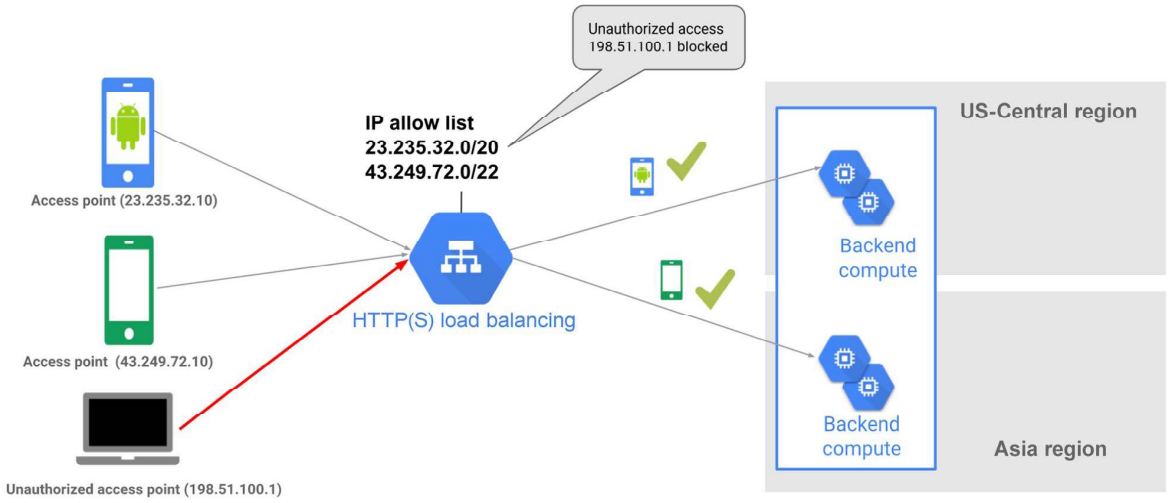

Figure 8.22 – Cloud Armor named IP list

In *Figure 8.22*, you can see how named IP lists work. CDN provides a list of IP addresses, such as `23.235.32.0/20`, `43.249.72.0/22`, and so on. Only communication coming from these IP addresses is permitted by this security rule. This means that the traffic of two CDN providers can be accessed (the access points are `23.235 32 10` and `43.249 72 10`), while unapproved traffic from `198.51.100.1` is banned.

Summary

In this chapter, we looked at some advanced network security concepts. We covered the usage and configuration of Private Google Access, IAP and its use cases, and Cloud NAT. Finally, we looked at Cloud Armor and how it provides protection against DDoS and web-application-based attacks. We also covered additional details related to Cloud Armor, such as security policies, WAF rules, and named IP lists.

In the next chapter, we will cover data security, which is an important topic for the exam. We will look at the Google Cloud Key Management system and then cover data loss prevention and Secret Manager in the subsequent chapters.

Further reading

For more information on Google Cloud advanced network security, refer to the following links:

- Internet NEGs: `https://packt.link/DlCXS`
- Zonal NEGs: `https://packt.link/IlB39`
- Tune Google Cloud Armor preconfigured WAF rules: `https://packt.link/YEScF`
- Configuring Private Google Access and Cloud NAT: `https://packt.link/JyutC`
- Securing Cloud Applications with Identity-Aware Proxy (IAP) using Zero-Trust: `https://packt.link/bM1OX`
- Rate Limiting with Cloud Armor: `https://packt.link/rmCAn`

9

Google Cloud Key Management Service

In this chapter, we will look at Google Cloud **Key Management Service** (**KMS**). Cloud KMS is a foundational service for all cryptographic operations in Google Cloud. Every workload that you deploy on Google Cloud is going to need the ability to encrypt data and use it for authorized purposes. There are various options presented by Cloud KMS, and it's essential to understand them and make an informed choice to help with regulatory and audit requirements.

In this chapter, we will cover the following topics:

- Overview of Cloud KMS
- Encryption and key management in Cloud KMS
- Key management options
- Customer-supplied encryption key
- Symmetric and asymmetric key encryption
- Bringing your own key to the cloud
- Key lifecycle management
- Key IAM permissions
- Cloud HSM
- Cloud External Key Manager
- Cloud KMS best practices
- Cloud KMS APIs and logging

Let us start by learning about the capabilities of Cloud KMS.

Overview of Cloud KMS

With Cloud KMS, Google's focus is to provide a scalable, reliable, and performant solution with a wide spectrum of options that you can control on a platform that is straightforward to use. Let us start with a quick overview of the Cloud KMS architecture.

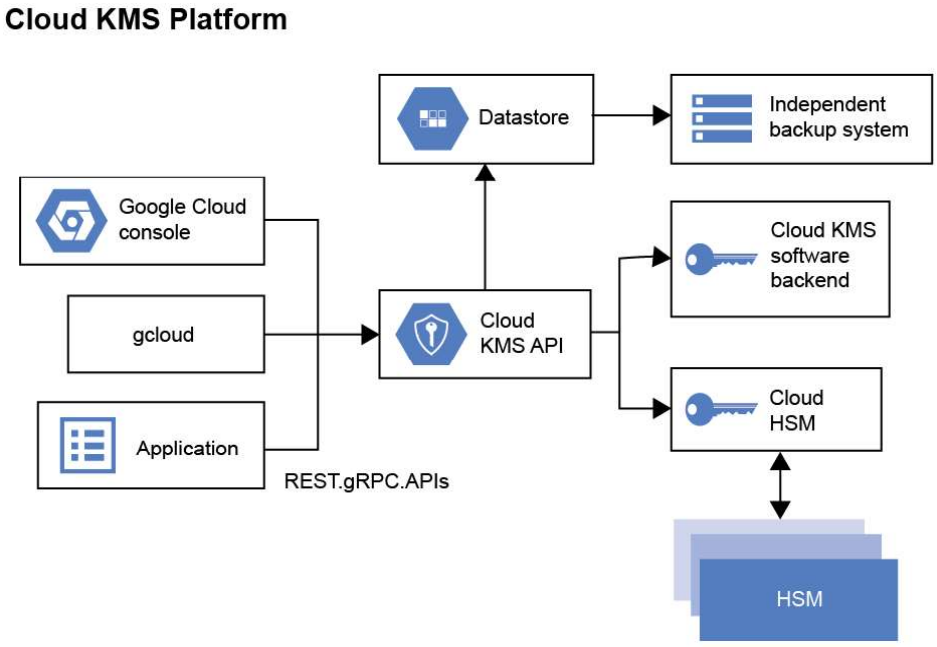

Figure 9.1 – The Cloud KMS architecture

The key components of the Cloud KMS platform are depicted in *Figure 9.1*. Administrators can access key management services through the Google Cloud console or CLI, as well as through the REST or gRPC APIs. A REST API or gRPC is used by applications to access key management services.

When creating a key on the Cloud KMS platform, you can select a protection level to define which key backend the key should use. The Cloud KMS platform has two backends (excluding Cloud EKM): the **software** and **HSM** protection levels. The software protection level is for keys that are protected by the **software security module**. HSM refers to keys that are protected by **hardware security modules** (**HSMs**).

Cloud KMS cryptographic operations are performed by FIPS 140-2-validated modules. Keys with the software protection level, and the cryptographic operations performed with them, comply with FIPS 140-2 Level 1. Keys with the HSM protection level, and the cryptographic operations performed with them, comply with FIPS 140-2 Level 3.

Let us now see the current offerings of Cloud KMS.

Current Cloud KMS encryption offerings

In this section, we will look at various KMS offerings. The offerings are based on the source of the key material, starting from Google-managed encryption and going up to where the key material is managed externally.

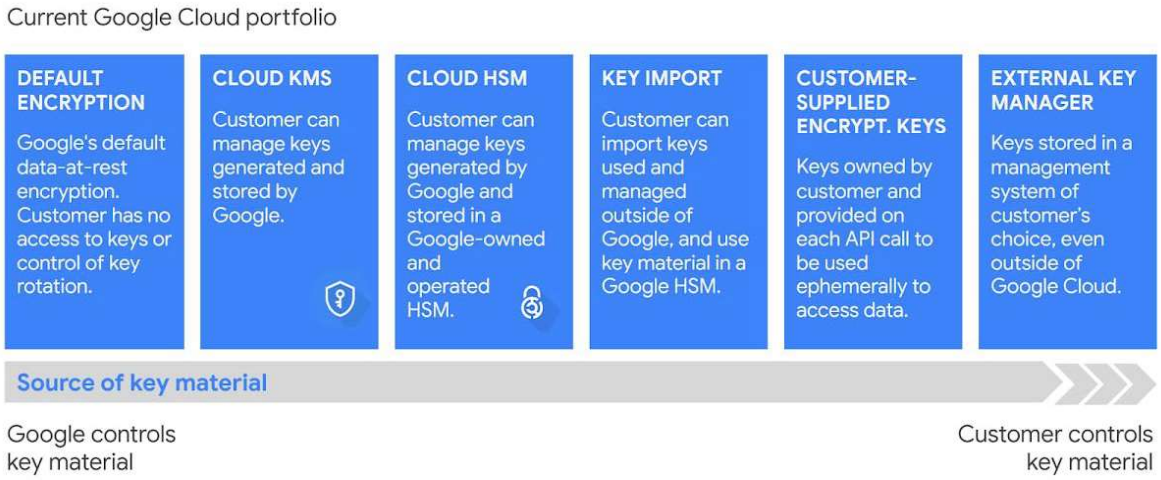

Figure 9.2 – Google Cloud encryption offerings

Figure 9.2 shows different offerings of encryption on Google Cloud.

Cloud KMS is supported by five design pillars:

1. **Customer control**: You can use Cloud KMS to handle software and hardware encryption keys, or you can provide your own via the key import functionality.
2. **Control and monitoring of access**: You can manage permissions on individual keys and see how they are used.
3. **Regionality**: Regionalization is built into Cloud KMS. Only the Google Cloud location you choose is used to produce, store, and process software keys in the service.
4. **Durability**: Cloud KMS meets the most stringent durability requirements. Cloud KMS checks and backs up all key material and metadata on a regular basis to protect against data corruption and ensure that data can be correctly decrypted.
5. **Security**: Cloud KMS is fully linked with **Identity and Access Management** (**IAM**) and **Cloud Audit Logs** controls, providing robust protections against unwanted access to keys. Cloud Audit Logs helps you understand who used a key and when.

Now that you understand the underlying principles of the Cloud KMS offerings, let us learn the basics of key management.

Encryption and key management in Cloud KMS

In this section, we will learn the basics of key management as it relates to Cloud KMS.

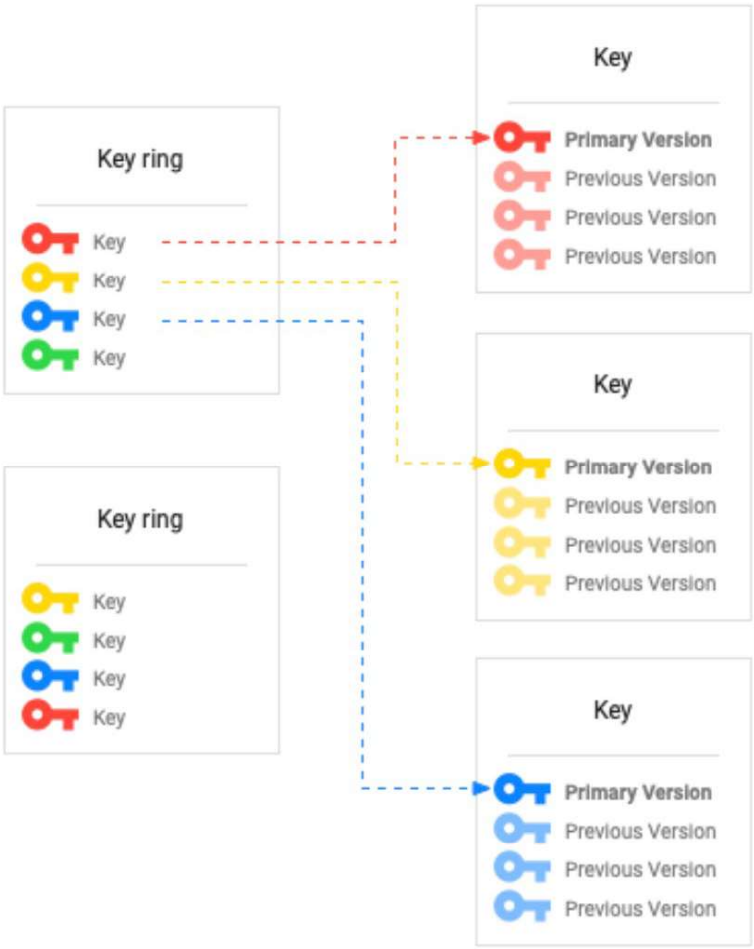

Figure 9.3 – Key structure

Figure 9.3 shows the structure of keys in Cloud KMS. Let us learn about the terms of key management:

- **Key**: A named object representing a cryptographic key that is used for a specific purpose. The key material—the actual bits used for cryptographic operations—can change over time as you create new key versions. Key purpose and other attributes of the key are connected with and managed using the key. Thus, the key is the most important object for understanding Cloud KMS usage. Cloud KMS supports both asymmetric keys and symmetric keys. A symmetric key is used for symmetric encryption to protect some corpus of data—for example, using AES-256 in GCM mode to encrypt a block of plaintext. An asymmetric key can be used for asymmetric encryption, or for creating digital signatures.

- **Key ring**: A grouping of keys for organizational purposes. A key ring belongs to a Google Cloud project and resides in a specific location. Keys inherit IAM policies from the key ring that contains them. Grouping keys with related permissions in a key ring lets you grant, revoke, or modify permissions to those keys at the key ring level without needing to act on each key individually. Key rings provide convenience and categorization, but if the grouping of key rings is not useful to you, you can manage permissions directly on keys. To prevent resource name collisions, a key ring cannot be deleted. Key rings and keys do not have billable costs or quota limitations, so their continued existence does not affect costs or production limits.
- **Key metadata**: Resource names, properties of Cloud KMS resources such as IAM policies, key type, key size, key state, and any data derived from them. Key metadata can be managed differently than the key material.
- **Key version**: Represents the key material associated with a key at some point in time. The key version is the resource that contains the actual key material. Versions are numbered sequentially, beginning with version 1. When a key is rotated, a new key version is created with new key material. The same logical key can have multiple versions over time, thus limiting the use of any single version. Symmetric keys will always have a primary version. This version is used for encrypting by default. When Cloud KMS performs decryption using symmetric keys, it automatically identifies which key version is needed to perform the decryption. A key version can only be used when it is enabled. Key versions in any state other than destroyed incur costs.
- **Purpose**: A key's purpose determines whether the key can be used for encryption or for signing. You choose the purpose when creating the key, and all versions have the same purpose. The purpose of a symmetric key is always symmetric encrypt/decrypt. The purpose of an asymmetric key is either asymmetric encrypt/decrypt or asymmetric signing. You create a key with a specific purpose in mind and it cannot be changed after the key is created.
- **State**: A key version's state is always one of the following:
  - Enabled
  - Disabled
  - Scheduled for destruction
  - Destroyed

Let us now move on to understand the key hierarchy in Cloud KMS.

Key hierarchy

The following diagram shows the key hierarchy of Cloud KMS. Cloud KMS uses Google's internal KMS in that Cloud-KMS-encrypted keys are wrapped by Google's KMS. Cloud KMS uses the same root of trust as Google's Cloud KMS.

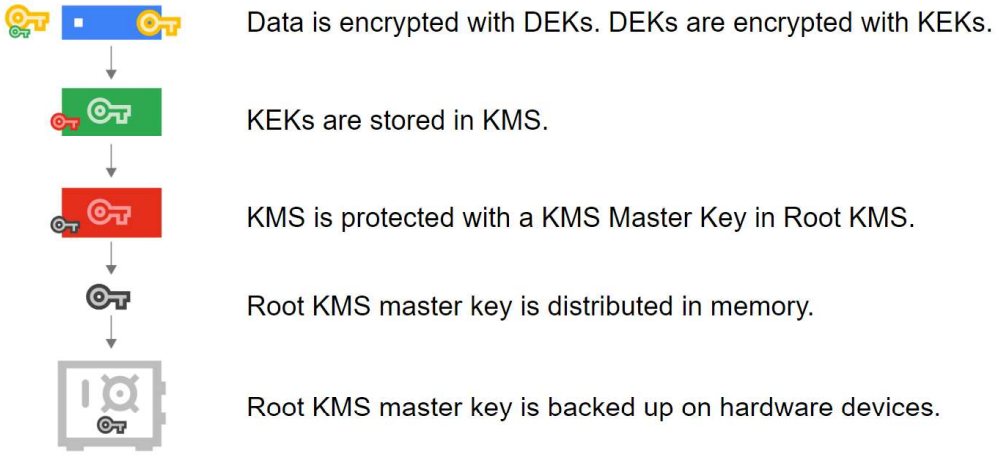

Figure 9.4 – Key hierarchy

Figure 9.4 shows the key hierarchy in Cloud KMS. There are several layers in the hierarchy:

- **Data encryption key** (**DEK**): A key used to encrypt actual data. DEKs are managed by a Google Cloud service on your behalf.
- **Key encryption key** (**KEK**): A key used to encrypt, or wrap, a data encryption key. All Cloud KMS platform options (software, hardware, and external backends) let you control the KEK.
- **KMS master key**: The key used to encrypt the KEK. This key is distributed in memory. The KMS master key is backed up on hardware devices. This key is responsible for encrypting KEKs.
- **Root KMS master key**: Google's internal KMS key.

Now that you understand how key hierarchy works, let us go over the basics of encryption.

Envelope encryption

As depicted in *Figure 9.5*, the process of encrypting data is to generate a DEK locally, encrypt data with the DEK, use a KEK to wrap the DEK, and then store the encrypted data and the wrapped DEK together. The KEK never leaves Cloud KMS. This is called **envelope encryption**. This section is more about understanding how Google handles envelope encryption, which is not something Google Cloud customers need to manage.

Figure 9.5 – Envelope encryption

Here are the steps that are followed to encrypt data using envelope encryption:

1. A DEK is generated locally, specifying a cipher type and a key material from which to generate the key.
2. This DEK is now used to encrypt the data.
3. A new KEK is either generated or an existing KEK from Cloud KMS is used—based on user preference. This KEK is used to encrypt (wrap) the DEK.
4. The wrapped DEK is stored with the encrypted data.

Now let us see the steps that are followed to decrypt data using envelope encryption.

As depicted in *Figure 9.6*, the process of decrypting data is to retrieve the encrypted data and the wrapped DEK, retrieve the KEK that wrapped the DEK, use the KEK to unwrap the DEK, and then use the unwrapped DEK to decrypt the data. The KEK never leaves Cloud KMS.

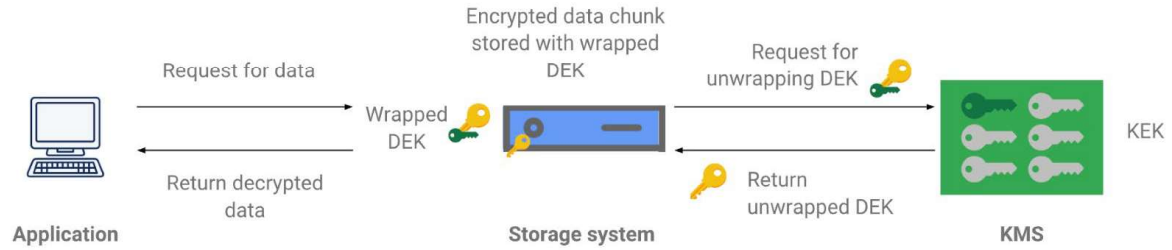

Figure 9.6 – Envelope decryption

To decrypt data using envelope encryption, you do the following:

1. Retrieve the encrypted data and the wrapped DEK.
2. Use the KEK stored in Cloud KMS to unwrap the encrypted DEK.
3. Use the plaintext DEK to decrypt the encrypted data.

Now that we understand the basics of keys and envelope encryption, let us look at key management options within Cloud KMS.

Key management options

In this section, we will cover several aspects of key management that are either Google-managed or customer-managed. It is important to know which option is best suited to which scenario so that you can make the right decision for a given use case.

Google Cloud's default encryption

Google Cloud stores all data encrypted at rest using a Google-managed default encryption key. The key is the AES-256 symmetric encryption key. There is no setup of keys or configuration required to turn on this option; all data by default uses this type of encryption. Google manages the keys and the rotation period of those keys. Google Cloud's default encryption is best suited for those customers who do not have specific requirements related to compliance or regional requirements for cryptographic key material. It is simple to use and does not require additional configuration to create keys, hence there is no cost to use it.

Customer-managed encryption keys (CMEKs)

Some customers of Google Cloud have stringent regulatory and compliance requirements to have control over the keys used to encrypt data at rest. To help with these requirements, Google Cloud offers the ability to encrypt data related to those services using an encryption key managed by the customer within Cloud KMS. CMEKs just mean that the customer decides the key type (software versus hardware), algorithm strength, and key rotation period when creating a key in Cloud KMS.

CMEKs give you control over aspects of the key lifecycle, including (but not limited to) the following:

- Customers can disable the keys used to safeguard data at rest, limiting Google's ability to decode it
- Customers can protect their data by utilizing a key that is personal to their location
- Customers can rotate the keys used to protect the data either automatically or manually
- Customers can use a Cloud HSM key, a Cloud **External Key Manager** (**EKM**) key, or an existing key that they import into Cloud KMS to protect their data

Customer-Managed Encryption Keys (**CMEKs**) are a supported feature in certain Google Cloud services. Let's explore what it entails for services to have CMEK support.

CMEK integrations

When a service supports CMEKs, it is said to have a CMEK integration. Some services, such as GKE, have multiple CMEK integrations for protecting several types of data related to the service.

For the exact steps to enable CMEKs, see the documentation for the relevant Google Cloud service. You can expect to follow steps like these:

1. You create or import a Cloud KMS key, selecting a location as geographically near as possible to the location of the service's resources. The service and the key can be in the same project or different projects. This is the CMEK.
2. You grant the CryptoKey Encrypter/Decrypter IAM role (`roles/cloudkms.cryptoKeyEncrypterDecrypter`) on the CMEK to the service account for that service.
3. You configure the service to use the CMEK to protect its data. For example, you can configure a GKE cluster to use a CMEK to protect data at rest on the boot disks of the nodes.

- If a service account has a `cryptoKeyEncrypterDecrypter` role, the service can encrypt and decrypt its data. If you revoke the role, or if you disable or destroy the CMEK, that data cannot be accessed.

> **Note**
> The rotation of the CMEK will not impact data availability since the service keeps track of which CMEK version was used to encrypt the data and will use that version to decrypt it.

CMEK compliance

Some services do not directly store data, or store data for only a brief period as an intermediate step in a long-running operation. For this type of workload, it is not practical to encrypt each write separately. These services do not offer CMEK integrations but can offer CMEK compliance, often with no configuration on your part.

A CMEK-compliant service encrypts temporary data by using an ephemeral key that only exists in memory and is never written to disk. When the temporary data is no longer needed, the ephemeral key is flushed from memory, and the encrypted data cannot be accessed, even if the storage resource still exists.

A CMEK-compliant service might offer the ability to send its output to a service with a CMEK integration, such as Cloud Storage.

Customer-supplied encryption key

You can provide your own AES-256 key, encoded in standard Base64, as an additional layer on top of Google-managed encryption keys. This key is known as a **customer-supplied encryption key** (**CSEK**).

The CSEK is currently supported by only two services: Cloud Storage and Compute Engine.

CSEKs with Cloud Storage

If you provide a CSEK, Cloud Storage does not permanently store your key on Google's servers or otherwise manage your key.

Instead, you provide your key for each Cloud Storage operation, and your key is purged from Google's servers after the operation is complete. Cloud Storage stores only a cryptographic hash of the key so that future requests can be validated against the hash. Your key cannot be recovered from this hash, and the hash cannot be used to decrypt your data. Here are the best practices when using CSEKs:

- You should back up each key to a secure location and take precautions to ensure that keys are not shared with untrusted parties
- If any file or machine containing the encryption key is compromised, you should immediately perform key rotation for all objects encrypted with the compromised key

When you apply a customer-supplied encryption key to an object, Cloud Storage uses the key when encrypting:

- The object's data
- The object's CRC32C checksum
- The object's MD5 hash

Cloud Storage uses standard server-side keys to encrypt the remaining metadata for the object, including the object's name. This allows you to read and update general metadata, as well as list and delete objects, without needing the CSEK. However, to perform any of these actions, you must have sufficient permission to do so.

For example, if an object is encrypted with a CSEK, the key must be used to perform operations on the object such as downloading or moving it. If you attempt to read the object's metadata without providing the key, you receive metadata, such as the object name and content type, but not the object's CRC32C checksum or MD5 hash. If you do supply your key with the request for the object metadata, the object's CRC32C checksum and MD5 hash are included with the metadata.

The following restrictions apply when using CSEKs:

- Cloud Storage Transfer Service and Cloud Dataflow do not currently support objects encrypted with CSEKs.
- You cannot use the Google Cloud console to download objects that are encrypted with a CSEK. Similarly, when you use the Google Cloud console to upload an object, you cannot encrypt the object with a CSEK.
- You can only set CSEKs on individual objects. You cannot set a default CSEK for a bucket.

- If you are performing a composition operation on objects encrypted by CSEKs, the component objects must be encrypted by the same key, and you need to provide the key with the compose request. The resulting composite object is encrypted by the same key.

Now that you understand the restrictions of using CSEKs with Google Cloud Storage, here is how you can use CSEKs with `gsutil`.

CSEKs with gsutil

To use a CSEK with `gsutil`, add the `encryption_key` parameter to the `[GSUtil]` section of your boto configuration file. `encryption_key` is an RFC 6848 Base64-encoded string of your AES-256 encryption key.

You can optionally specify one or more decryption keys, up to 100. While the `encryption_key` option is used by `gsutil` as both an encryption and decryption key, any `decryption_key` options you specify are used only to decrypt objects. Multiple decryption keys must be listed in the boto configuration file as follows:

```
decryption_key1 = ...
decryption_key2 = ...
decryption_key3 = ...
```

If you have encryption or decryption keys in your boto configuration file, they are used for all `gsutil` commands. When decrypting, `gsutil` calculates the SHA256 hash of the supplied encryption and decryption keys and selects the correct key to use for a particular object by matching the SHA256 hash in the object's metadata.

Objects encrypted with CSEKs require the matching decryption key during **encryption key rotation**. If an object is encrypted using a CSEK, you can rotate the object's key by rewriting the object. Rewrites are supported through the JSON API, but not the XML API.

Now that we have seen how CSEKs are used with Cloud Storage, let us see how they can be used with Compute Engine.

CSEKs with Compute Engine

Compute Engine encrypts all data at rest by default on the persistent disks by using Google default encryption keys. You can bring your own encryption key if you want to control and manage this encryption yourself.

If you provide your own encryption key, Compute Engine uses your key to protect the Google-generated keys used to encrypt and decrypt your data. Only users who can provide the correct key can use resources protected by a CSEK.

Google does not store your keys on its servers, and it cannot access your protected data unless you provide it to them. This also implies that if you forget or lose your key, Google will be unable to recover it, as well as any data encrypted with it.

Google discards the cryptographic keys when you destroy a persistent disk, making the data unrecoverable. This is an irreversible procedure.

There are some technical restrictions in using CSEKs with Compute Engine:

- You can only encrypt new persistent disks with your own key. You cannot encrypt existing persistent disks with your own key.
- You cannot use your own keys with local SSDs because local SSDs do not persist beyond the life of a VM. Local SSDs are already protected with an ephemeral encryption key that Google does not retain.
- Compute Engine does not store encryption keys with instance templates, so you need to store your own keys in Cloud KMS to encrypt disks in a managed instance group.

Now you understand the restrictions of using CSEKs with Compute Engine, please look at the Google Cloud documentation on how to use CSEKs with Compute Engine. Try out a few of these use cases on your own to get a fair idea of how a CSEK is used:

- Encrypt a new persistent disk with a CSEK
- Create a snapshot from a disk encrypted with a CSEK
- Create a new image from a disk or custom image encrypted with a CSEK
- Encrypt an imported image with a CSEK
- Create a persistent disk from a resource encrypted with a CSEK
- Attach a disk encrypted with a CSEK to a new VM
- Start or restart VMs that have disks encrypted with a CSEK

So far, we have seen different types of keys and where you can use them. Let us move on now to understand symmetric and asymmetric key operations.

Symmetric key encryption

Recall that symmetric keys are used for encryption to protect some data—for example, using AES-256 in GCM mode to encrypt a block of plaintext.

Creating a symmetric key

To create a symmetric key, you will first need to create a key ring. The key ring determines the location of the key. Let us start with creating that.

Step 1: Creating a key ring

Here is a `gcloud` command to create a key ring:

```
gcloud kms keyrings create key-ring-name \
    --location location
```

Replace `key-ring-name` with a name for the key ring to hold the key. Replace `location` with the Cloud KMS location for the key ring and its keys.

Step 2: Creating a key

Use the following command to create a key in an existing key ring:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "encryption"
```

Replace `key` with the name of the key you would like to use and `key-ring-name` with the name of the key ring you created in the previous step. Replace `location` with the Cloud KMS location of the key ring. Note the *purpose* of this key.

Step 3: Setting a key rotation period and starting time

Once the key is created, you can set a rotation period with a starting time. Here is the command that will allow you to do so:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "encryption" \
    --rotation-period rotation-period \
    --next-rotation-time next-rotation-time
```

Replace `key` with a name for the key. Replace `key-ring-name` with the name of the existing key ring where the key will be located. Replace `location` with the Cloud KMS location for the key ring. Replace `rotation-period` with an interval, such as `30d` to rotate the key every 30 days. Replace `next-rotation-time` with a timestamp at which to begin the first rotation, such as `1970-01-01T01:02:03`.

Now that you have created a key for symmetric encryption, let us see how to use it to encrypt content.

Encrypting content with a symmetric key

Here is a command to use an existing key for encryption:

```
gcloud kms encrypt \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --plaintext-file file-with-data-to-encrypt \
    --ciphertext-file file-to-store-encrypted-data
```

Replace `key` with the name of the key to use for encryption. Replace `key-ring-name` with the name of the key ring where the key is located. Replace `location` with the Cloud KMS location for the key ring. Replace `file-with-data-to-encrypt` and `file-to-store-encrypted-data` with the local file paths (include the actual filename as well) for reading the plaintext data and saving the encrypted output.

This command will produce a file containing encrypted data. The name of the file is the one you provided for the `ciphertext-file` argument.

Now that you have seen how to encrypt the content, let us see how to decrypt it.

Decrypting content with a symmetric key

Here is a command to use an existing key for decryption:

```
gcloud kms decrypt \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --ciphertext-file file-path-with-encrypted-data \
    --plaintext-file file-path-to-store-plaintext
```

Replace `key` with the name of the key to use for encryption. Replace `key-ring-name` with the name of the key ring where the key is located. Replace `location` with the Cloud KMS location for the key ring. Replace `file-path-with-encrypted-data` and `file-path-to-store-plaintext` with the local file paths (include the actual filename as well) for reading the plaintext data and saving the encrypted output.

This command will produce a file containing decrypted data. The name of the file is the one you have provided for `plaintext-file`.

Now that you have seen symmetric key encryption and decryption using Cloud KMS, let us move on to see how to use asymmetric key encryption.

Asymmetric key encryption

The following section describes the flow for using an asymmetric key to encrypt and decrypt data. Asymmetric key encryptions involve a key pair (public and private key pair). As the name suggests, the private key is not shared while the public key is shared. There are two participants in this workflow—a sender and a recipient. The sender creates a ciphertext using the recipient's public key, and then the recipient decrypts the ciphertext using the private key it holds. Only someone with knowledge of the private key can decrypt the ciphertext.

Cloud KMS provides the following functionality as it relates to asymmetric encryption:

- The ability to create an asymmetric key with the key purpose of `ASYMMETRIC_DECRYPT`. For information about which algorithms Cloud KMS supports, see asymmetric encryption algorithms in the Google Cloud documentation.
- CloudKMS asymmetric keys also support `ASYMMETRIC_SIGN` (ECC and RSA).
- The ability to retrieve the public key for an asymmetric key. You use the public key to encrypt data. Cloud KMS does not directly provide a method to asymmetrically encrypt data. Instead, you encrypt data using openly available SDKs and tools, such as `OpenSSL`. These SDKs and tools require the public key that you retrieve from Cloud KMS.
- The ability to decrypt data with an asymmetric key.

Now let us look at the steps to be able to do asymmetric encryption and signing. Similar to symmetric key creation, in order to create an asymmetric key, you will first need to create a key ring. The key ring determines the location of the key. Let us start.

Step 1: Creating a key ring

Use the following command to create a key ring for your asymmetric key:

```
gcloud kms keyrings create key-ring-name \
    --location location
```

Replace `key-ring-name` with a name for the key ring. Replace `location` with the Cloud KMS location for the key ring and its keys.

Step 2: Creating an asymmetric decryption key

Follow these steps to create an asymmetric decryption key on the specified key ring and location. These examples use the software protection level and an `rsa-decrypt-oaep-2048-sha256` algorithm. When you first create the key, the key's initial version has a state of pending generation. When the state changes to enabled, you can use the key:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "asymmetric-encryption" \
    --default-algorithm "rsa-decrypt-oaep-2048-sha256"
```

Replace `key` with a name for the new key. Replace `key-ring-name` with the name of the existing key ring where the key will be located. Replace `location` with the Cloud KMS location for the key ring.

Step 3: (Optional) Creating an asymmetric signing key

Follow these steps to create an asymmetric signing key on the specified key ring and location. The following command uses the software protection level and the `rsa-sign-pkcs1-2048-sha256` algorithm. When you first create the key, the key's initial version has a state of pending generation. When the state changes to enabled, you can use the key:

```
gcloud kms keys create key \
    --keyring key-ring-name \
    --location location \
    --purpose "asymmetric-signing" \
    --default-algorithm "rsa-sign-pkcs1-2048-sha256"
```

Now let us move on to understand how to use an asymmetric key.

Encrypting data with an asymmetric key

This section provides examples that run at the command line. We use OpenSSL to encrypt data. Cloud Shell is pre-installed with OpenSSL, so go ahead and try this in Cloud Shell.

> **Note**
>
> You cannot follow this procedure with a key that has a purpose of `ASYMMETRIC_SIGN`.
>
> The version of OpenSSL installed on macOS does not support the flags used to decrypt data in this topic. To follow these steps on macOS, install OpenSSL from Homebrew.

Step 1: Downloading the public key

Run the following command to download the public key part of the asymmetric key that you created in *Step 2* in the previous section:

```
gcloud kms keys versions get-public-key key-version \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --output-file public-key-path
```

Replace `key-version` with the key version that has the public key. Replace `key` with the name of the key. Replace `key-ring-name` with the name of the key ring where the key is located. Replace `location` with the Cloud KMS location for the key ring. Replace `public-key-path` with the location to save the public key on the local system.

Now encrypt the data with the public key you downloaded from Cloud KMS:

```
openssl pkeyutl -in cleartext-data-input-file \
    -encrypt \
    -pubin \
    -inkey public-key-path \
    -pkeyopt rsa_padding_mode:oaep \
    -pkeyopt rsa_oaep_md:sha256 \
    -pkeyopt rsa_mgf1_md:sha256 \
    > encrypted-data-output-file
```

Replace `cleartext-data-input-file` with the path and filename to encrypt. Replace `public-key-path` with the path and filename where you downloaded the public key. Replace `encrypted-data-output-file` with the path and filename to save the encrypted data.

Now let us see how to decrypt the data using an asymmetric key.

Decrypting data with an asymmetric key

We will use the private key part of the asymmetric key to decrypt the encrypted data. Run the `gcloud` command for decryption:

```
gcloud kms asymmetric-decrypt \
    --version key-version \
    --key key \
    --keyring key-ring-name \
    --location location  \
    --ciphertext-file file-path-with-encrypted-data \
    --plaintext-file file-path-to-store-plaintext
```

Replace `key-version` with the key version or omit the `--version` flag to detect the version automatically. Replace `key` with the name of the key to use for decryption. Replace `key-ring-name` with the name of the key ring where the key will be located. Replace `location` with the Cloud KMS location for the key ring. Replace `file-path-with-encrypted-data` and `file-path-to-store-plaintext` with the local file paths for reading the encrypted data and saving the decrypted output, respectively.

The decrypted text will be in `plaintext-file`.

We looked at some common operations with a key generated by KMS. Now let us look at an advanced option to bring your own key to the cloud, which some organizations may prefer to achieve compliance.

Importing a key (BYOK)

Google allows you to bring your own cryptographic key material. You can import that using the Software or Cloud HSM protection level. We will see step-by-step instructions on how to do this. But before we do that, let us understand the reasons you want to import a key:

- You may be using existing cryptographic keys that were created on your premises or in an external KMS.
- If you migrate an application to Google Cloud or if you add cryptographic support to an existing Google Cloud application, you can import the relevant keys into Cloud KMS.
- As part of key import, Cloud KMS generates a wrapping key, which is a public/private key pair, using one of the supported import methods. Encrypting your key material with this wrapping key protects the key material in transit.
- This Cloud KMS public wrapping key is used to *encrypt*, on the client, the key to be imported. The private key matching this public key is stored within Google Cloud and is used to unwrap the key after it reaches the Google Cloud project. The import method you choose determines the algorithm used to create the wrapping key.

Before we begin to go over instructions on how to bring your key, make sure of the following prerequisites:

- Prepare the local system by choosing *one* of the following options. Automatic key wrapping is recommended for most users.
    - **If you want to allow gcloud to wrap your keys automatically** before transmitting them to Google Cloud, you must install the Pyca cryptography library on your local system. The Pyca library is used by the import job that wraps and protects the key locally before sending it to Google Cloud.
    - **If you want to wrap your keys manually**, you must configure OpenSSL for manual key wrapping.

- In addition to these options, make sure you follow these guidelines as well:
  - Verify that your key's algorithm and length are supported by Google Cloud KMS. Allowable algorithms for a key depend on whether the key is used for symmetric encryption, asymmetric encryption, or asymmetric signing, and whether it is stored in software or an HSM. You will specify the key's algorithm as part of the import request.
  - Separately, you must also verify how the key is encoded, and adjust if necessary.

Now let us start with the instructions on importing keys to Google Cloud. For simplicity, we will be using `gcloud`.

Step 1: Creating a blank key

1. Create a project in the Google Cloud console, and enable Cloud KMS.
2. If you do not have a key to import but want to test importing keys, you can create a symmetric key on the local system, using the following command:

```
openssl rand 32 > ${HOME}/test.bin
```

> **Note**
> Use this key for testing only. A key created this way might not be appropriate for production use.

3. Create a new key ring in a region of your choice.
4. Create a new key of type imported key by using the following parameters:
   I. **Name**: `ext-key-3`
   II. **Protection Level**: `Software`
   III. **Purpose**: `Symmetric encrypt/decrypt`
   IV. **Rotation Period**: `Manual`
5. An empty key is created with no version; you will be redirected to create a key version.
6. Use the following `gcloud` command to create the key version. We will be creating a symmetric key:

```
gcloud kms keys create ext-key-4 --location us-central1
--keyring external-central-key-ring-name --purpose encryption
--skip-initial-version-creation
```

7. Create a new import job.

8. Download the wrapping key.

   This key will not be available for use yet since it is an empty shell to hold the key material that you will import in the following steps. Now, you have two options:

   - Upload your pre-wrapped encryption key.
   - Use the `gcloud` command to provide your key to the job, which will automatically wrap it. You will use different flags to make the import request, depending on whether you want the `gcloud` command-line tool to wrap your key automatically or you have wrapped your key manually. We will use the `gcloud` option here as it is simple to understand.

Step 2: Importing the key using an import job

1. Set `export CLOUDSDK_PYTHON_SITEPACKAGES=1`, otherwise, you will see the following error:

   ```
   Cannot load the Pyca cryptography library. Either the library is
   not installed, or site packages are not enabled for the Google
   Cloud SDK. Please consult https://cloud.google.com/kms/docs/
   crypto for further instructions
   ```

2. Make sure the preceding environment variable is set by using this command:

   ```
   echo $CLOUDSDK_PYTHON_SITEPACKAGES
   ```

3. Execute the following command to upload your key to Cloud KMS:

   ```
   user@cloudshell:~ $ gcloud kms keys versions import   --import-
   job ext-key-import-job   --location us-central1   --keyring
   external-central-key-ring-name   --key ext-key-4   --algorithm
   google-symmetric-encryption   --target-key-file ${HOME}/test.bin
   --public-key-file /home/user/ext-key-import-job.pub --verbosity
   info
   ```

   You will see an output similar to what is shown here:

   ```
   INFO: Display format: "default"
   algorithm: GOOGLE_SYMMETRIC_ENCRYPTION
   createTime: '2020-04-08T22:25:46.597631370Z'
   importJob: projects/PROJECT_ID/locations/us-central1/keyRings/
   external-central-key-ring-name/importJobs/ext-key-import-job
   name: projects/PROJECT_ID/locations/us-central1/keyRings/
   external-central-key-ring-name/cryptoKeys/ext-key-1/
   cryptoKeyVersions/1
   protectionLevel: SOFTWARE
   state: PENDING_IMPORT
   ```

4. Wait until the key version is in the **ENABLED** state, then set it to primary using the following `gcloud` command:

```
gloud kms keys set-primary-version ext-key-1 --version=1
--keyring=external-central-key-ring-name --location=us-central1
```

Cloud KMS will only use the primary version of the key. Now that you have set it, let us move on to using it.

Step 3: Verifying key encryption and decryption

1. Create a `secrets.txt` file with some plain text in it. Encrypt that file using `gcloud`:

```
 gcloud kms encrypt --ciphertext-file=secrets.txt.enc
--plaintext-file=secrets.txt --key=ext-key-4 --keyring=external-
central-key-ring-name --location=us-central1
```

2. Now use the following command to decrypt it. Decrypt the file using the same key:

```
gcloud kms decrypt --ciphertext-file=secrets.txt.
enc --plaintext-file=secrets.txt.dec --key=ext-key-1
--keyring=external-central-key-ring-name --location=us-central1
```

   Verify the text you have decrypted is the same as the plain text. This tutorial demonstrated that you can bring your own key material and use it for encryption and decryption.

Now that we understand several types of keys you can use in Cloud KMS, let us understand the lifecycle of keys.

Key lifecycle management

While operating your workloads, you need to manage the lifecycle of your keys. The US **National Information Standards Institute** (**NIST**) **special publication** (**SP**) *800-57, Part 1* describes a key management lifecycle that is divided into four phases: pre-operational, operational, post-operational, and destroyed.

The following section provides a mapping of the NIST lifecycle functions from the publication to Google Cloud KMS lifecycle functions.

1. The **Pre-operational** lifecycle phase is mapped to the following:

   I. **NIST section 8.1.4 Keying-Material Installation Function**: The equivalent Cloud KMS operation is **Key import**.

   II. **NIST section 8.1.5 Key Establishment Function**: The equivalent Cloud KMS operation is **Key creation (symmetric, asymmetric)**.

2. The **Operational** lifecycle phase is mapped to the following:

    I. **NIST section 8.2.1 Normal Operational Storage Function**: The equivalent Cloud KMS operation is **Key creation in SOFTWARE, HSM, or EXTERNAL protection levels (symmetric, asymmetric)**.

    II. **NIST section 8.2.3 Key Change Function**: The equivalent Cloud KMS operation is **Key rotation**.

3. The **Post-operational** lifecycle phase is mapped to the following:

    I. **NIST section 8.3.4 Key Destruction Function**: The equivalent Cloud KMS operation is **Key destruction (and recovery)**.

In addition, Cloud KMS offers the capability to disable a key. Typically, you disable the old key after rotating to a new key. This gives you a period to confirm that no additional data needs to be re-encrypted before the key is destroyed. You can re-enable a disabled key if you discover data that needs to be re-encrypted. When you are sure that there is no more data that was encrypted by using the old key, the key can be scheduled for destruction.

Key IAM permissions

When considering key permissions, think about the hierarchy in which the key exists. A key exists in a key ring, a project, a folder in another folder, or under "Cloud Organization".

Recall that there are two fundamental security principles IAM enforces:

- Principle of separation of duties
- Principle of least privilege

A primary role a principal can play is the Cloud KMS CryptoKey Encrypter/Decrypter role at various levels of the hierarchy. There are several other roles Cloud KMS has based on how you structure it. Please refer to the Google Cloud KMS documentation for the list of other IAM roles: `https://packt.link/RyY17`.

We have looked at various IAM roles and permissions for Cloud KMS; let us now look at some best practices on how to manage access:

- Key management roles can be granted based on the culture and process of your enterprises. Traditionally, this role is played by the IT security team.
- For a large or complex organization, you might decide on an approach such as the following:

  - Grant members of your IT security team the Cloud KMS Admin role (`roles/cloudkms.admin`) across all projects. If different team members handle various aspects of a key's lifecycle, you can grant those team members a more granular role, such as the Cloud KMS Importer role (`roles/cloudkms.importer`).
  - Grant the Cloud KMS Encrypter/Decrypter role (`roles/cloudkms.cryptoKeyEncrypterDecrypter`) to users or service accounts that read or write encrypted data.
  - Grant the Cloud KMS Public Key Viewer role (`roles/cloudkms.publicKeyViewer`) to users or service accounts that need to view the public portion of a key used for asymmetric encryption.

- For a small organization, each project team can manage its own Cloud KMS and grant the IT security team access to audit the keys and usage.
- Consider hosting your keys in a separate Google Cloud project from the data protected by those keys. A user with a basic or highly privileged role in one project, such as an editor, cannot use this role to gain unauthorized access to keys in a different project.
- Avoid granting the owner role to any member. Without the owner role, no member in the project can create a key and use it to decrypt data or for signing, unless each of these permissions is granted to that member. To grant broad administrative access without granting the ability to encrypt or decrypt, grant the Cloud KMS Admin (`roles/cloudkms.admin`) role instead.
- To limit access to encrypted data, such as customer data, you can restrict who can access the key and who can use the key for decryption. If necessary, you can create granular custom roles to meet your business requirements.
- For projects using the VPC Service Controls perimeter, we recommend that you have a separate Cloud KMS project for the perimeter.

Now that we have looked at various IAM roles and best practices for key management, let us look at Cloud HSM offerings next.

Cloud HSM

Cloud HSM is a cloud-hosted HSM service that allows you to host encryption keys and perform cryptographic operations in a cluster of FIPS 140-2 Level 3 certified HSMs. The Cloud HSM cluster is managed by Google for you; you do not get access to the physical device. You also do not need to patch or scale it for multi-regional applications. Cloud HSM uses Cloud KMS as its frontend; you use Cloud KMS APIs to interact with the Cloud HSM backend. This abstracts the communication with Cloud HSM, so you do not need to use Cloud HSM-specific code.

When you use HSM-backed keys and key versions, the Google Cloud project that makes the cryptographic request incurs cryptographic operation quota usage, and the Google Cloud project that contains the HSM keys incurs HSM QPM quota usage.

> **Note**
> You can find more information on Cloud KMS in the Google Cloud documentation for quotas.

Here are some architectural characteristics of Cloud HSM that you should be aware of:

- The Cloud HSM service provides hardware-backed keys to Cloud KMS. It offers Google Cloud customers the ability to manage and use cryptographic keys that are protected by fully managed HSMs in Google data centers.
- The service is available and auto-scales horizontally. It is designed to be extremely resilient to the unpredictable demands of workloads.
- Keys are cryptographically bound to the Cloud KMS region in which the key ring was defined. You can choose to make keys global or enforce strict regional restrictions.
- With Cloud HSM, the keys that you create and use cannot be exported outside of the cluster of HSMs belonging to the region specified at the time of key creation.
- Using Cloud HSM, you can verifiably attest that your cryptographic keys are created and used exclusively within a hardware device.
- No application changes are required for existing Cloud KMS customers to use Cloud HSM: the Cloud HSM services are accessed using the same API and client libraries as the Cloud KMS software backend.
- If you choose to, you can use a PKCS#11-compliant C++ library, `libkmsp11to.so`, to interact with Cloud HSM from your application. You use this library typically for the following purposes:
  - Creating signatures, certifications, or **certificate signing requests** (**CSRs**) using the command line
  - TLS web session signing in web servers such as Apache or NGINX backed by Cloud HSM keys
  - Migrating your application to a Google Cloud project that uses PKCS#11 APIs

  This library authenticates Google Cloud using a service account. The library uses a YAML file to store the Cloud KMS configurations. Use the following link to learn more about this: `https://packt.link/H53BX`.

Let us look at the HSM key hierarchy now.