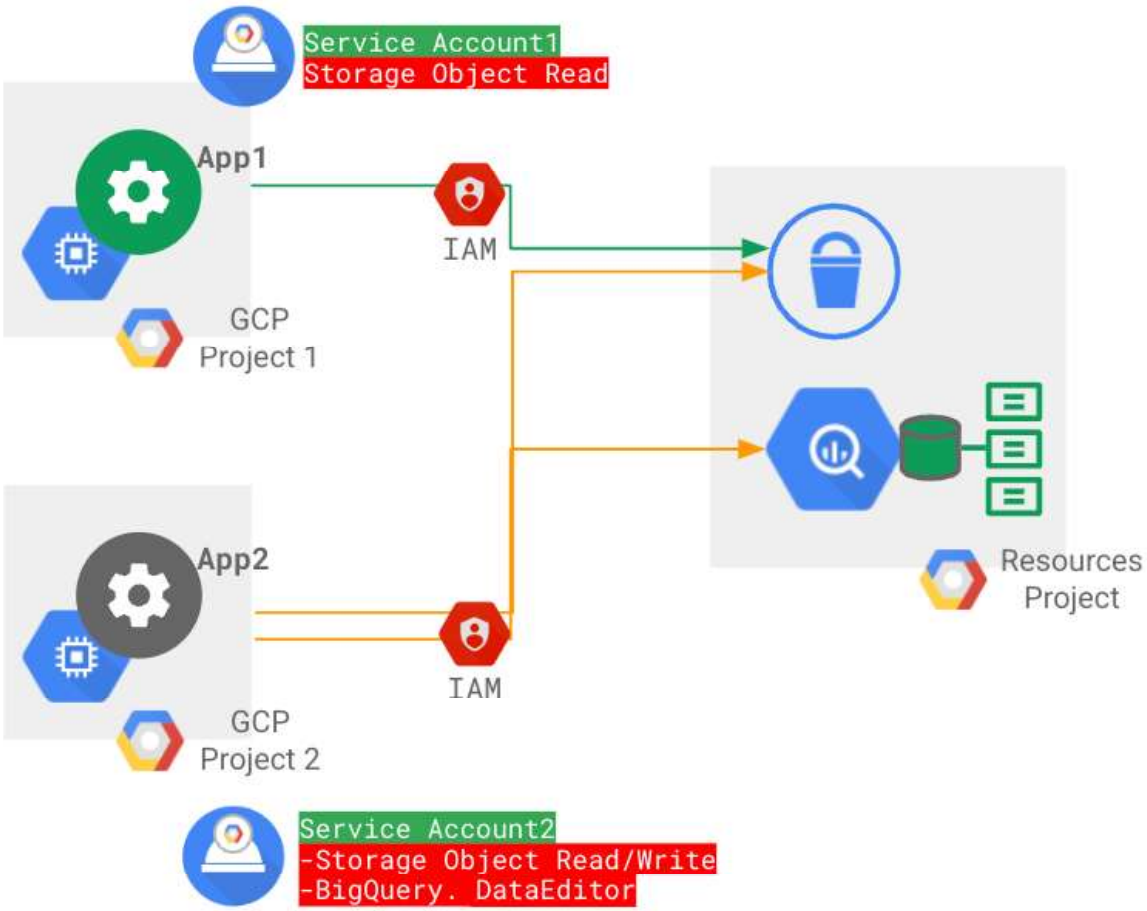

Figure 6.5 – Cross-project service account access

As depicted in *Figure 6.5*, to support this, Google Cloud has introduced the ability to allow access to service accounts to resources across projects. The default behavior is you can create a service account in one project and attach it to a resource in another project. If you do not want this behavior, you can set an organization policy to prevent it.

In the organization policy for the project where your service accounts are located, check the following Boolean constraints:

- Make sure that the `iam.disableCrossProjectServiceAccountUsage` Boolean constraint is not enforced for the project. When this constraint is not enforced, Cloud IAM adds a project lien that prevents the project from being deleted. This lien has the origin `iam.googleapis.com/cross-project-service-accounts`. It is strongly recommended that you do not delete this lien.

- It is recommended that the `iam.restrictCrossProjectServiceAccountLienRemoval` Boolean constraint is enforced for the project. This Boolean constraint ensures that principals can remove the project lien only if they have the `resourcemanager.projects.updateLiens` permission at the organization level. If this constraint is not enforced, principals can remove the project lien if they have this permission at the project level.

Before you take on enabling this feature, it is recommended that you create a proper design and understand all the dependencies that this process may bring across the enterprise, and make sure that you have considered the pros and cons of such a decision. After you update the organization policy, it is strongly recommended to not change it, especially in the production environment, as your Google Cloud resources might not work correctly.

Before we move on to service account activity, let us quickly go over how to configure WIF.

Configuring Workload Identity Federation with Okta

WIF enables access to Google Cloud services without the need for service account keys, thereby allowing access from on-premises or other cloud providers. In the following configuration, we assume the use of a third-party OIDC provider such as Okta for connecting to Google Cloud services. The client credentials flow, as defined in OAuth 2.0 RFC 6749, will be utilized. This flow requires the application to authenticate itself by passing the client ID and client Secret, after which a token is obtained. **Machine-to-Machine** (**M2M**) applications such as CLIs, batch jobs, or backend services authenticate and authorize as the application itself rather than a user. In such scenarios, typical authentication schemes such as username + password or social logins are not applicable. Instead, the OAuth client credentials flow is used.

> **Note**
>
> If your application requires a user context, consider using OAuth2.0 to access Google APIs on behalf of the user rather than WIF. You can find the details at `https://packt.link/UycpJ`.

Before we dive deeper into configurations, let us look at the data flow that happens in this scenario:

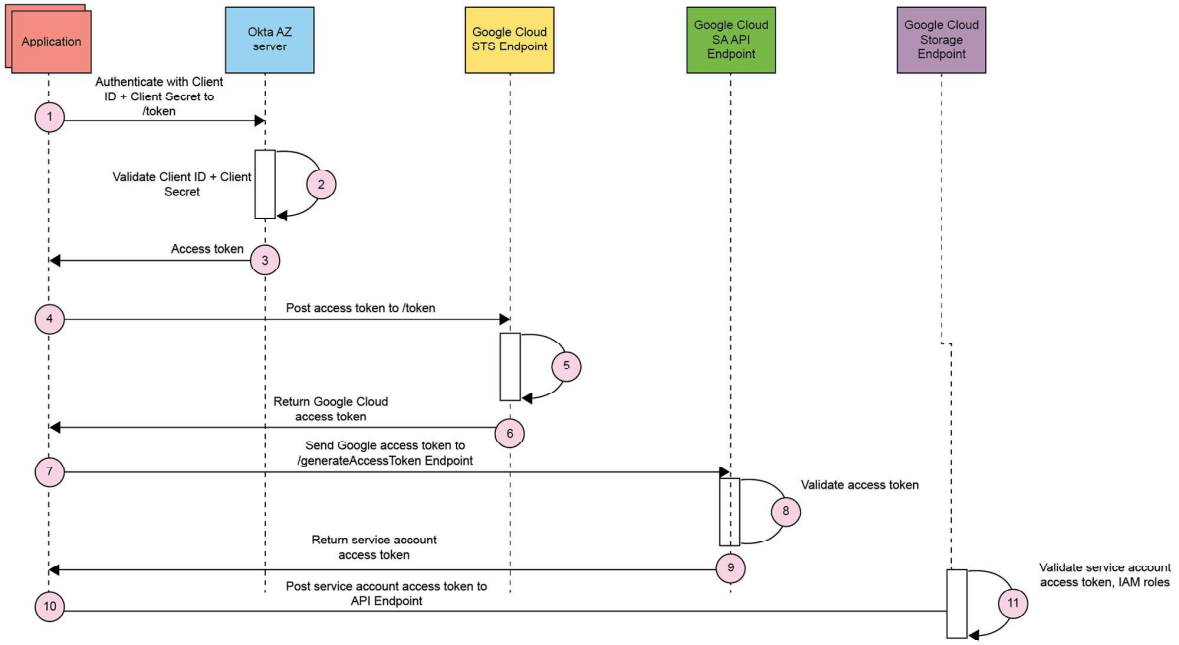

Figure 6.6 – Data flow for WIF with Okta

In the data flow shown in *Figure 6.6*, you will notice a call is made to the Okta authorization server using the client credentials to get an OIDC token. That token is then passed on to two Google Cloud endpoints before an access token can be obtained to call a Google Cloud service. Now let us go over the steps required to configure WIF:

1. Generate client credentials from Okta. Please refer to the Okta documentation on how to generate client credentials if you are not familiar with the process.
2. Create a workload identity pool using the following `gcloud` command:

```
gcloud iam workload-identity-pools create workload-id-pool1
— location="global" — description="Testing workload identity
federation with Okta " — display-name="workload-id-pool1"
```

3. Create an OIDC provider for the pool we just created:

```
gcloud iam workload-identity-pools providers create-oidc
okta-provider — workload-identity-pool="workload-id-pool1"
— issuer-uri="<okta-issuer-url>" — location="global" —
attribute-mapping="google.subject=assertion.sub" — allowed-
audiences="<ENTER CLIENT_ID>"
```

4. Add an IAM policy binding for the service account:

```
gcloud iam service-accounts add-iam-policy-binding
oktagcpsvacct@PROJECT_ID.iam.gserviceaccount.com —
role=roles/iam.workloadIdentityUser — member="principal://
iam.googleapis.com/projects/731056981369/locations/global/
workloadIdentityPools/workload-id-pool-1/subject/<authz-server-
sub>"
```

5. Update the provider with the allowed audience value as follows:

```
gcloud beta iam workload-identity-pools providers update-oidc
okta-provider — allowed-audiences="api://24wwds23" — workload-
identity-pool="workload-id-pool1" — location="global"
```

6. Use the following `gcloud` command to verify that the `aud` and `sub` values are set up correctly:

```
gcloud iam workload-identity-pools providers describe okta-
provider —workload-identity-pool="workload-id-pool1" —
location="global"
```

7. Now, download the JSON credentials configuration file and use it where you used the service account key before.

Service accounts pose a large threat vector to your Google Cloud organization, so it is pertinent that you monitor service account activity. In the next section, we will look at how to go about doing this.

Best practices for monitoring service account activity

It is important to understand service account activity to make sure that there is no threat actor or misuse of service accounts. Google Cloud provides Cloud Monitoring to view usage metrics for service accounts and service account keys.

These metrics will let you view and track usage patterns, which can help you identify anomalies.

Google APIs (for example, Google Maps and YouTube) that are not part of Google Cloud are included in these metrics if service accounts and service account credentials are used to call them. There are analytics for both successful and unsuccessful API requests. Using this example, the service account or key that was used for an API call failed because the caller was not permitted or because the request referenced a resource that does not appear in the metrics.

In these metrics, even if the system does not utilize the key to authenticate the request, service account keys appear if a system lists them while attempting to authenticate a request. Using signed URLs for Cloud Storage or authenticating third-party applications is the most prevalent cause of this problem. Usage statistics for non-authentication key pairs are therefore available.

To use Metrics Explorer to view the metrics for a monitored resource, follow these steps:

1. In the Google Cloud console, go to the **Metrics Explorer** page within **Monitoring**.
2. Go to **Metrics Explorer**.
3. In the toolbar, select the **Explorer** tab.
4. Select the **Configuration** tab.
5. Expand the **Select a metric** menu, enter `IAM Service Account` in the filter bar, and then use the submenus to select a specific resource type and metric (turn off **Show only active resources and metrics**):
    I. In the **Active resources** menu, select **IAM Service Account**.
    II. In the **Active metric categories** menu, select **Service_account**.
6. In the **Active metrics** menu, select a service account metric. The following metrics are available within your selected time interval:
    I. For service account usage metrics, select **Service account authentication events**.
    II. For service account key usage metrics, select **Service account key authentication events**.
7. Click **Apply**.
8. *Optional*: To configure how the data is viewed, use the **Filter**, **Group By, Aggregator**, and chart type menus.
9. *Optional*: Change the graph settings:
    I. For quota and other metrics that report one sample per day, set the period to at least one week and set the plot type to **Stacked bar chart**.
    II. For distribution-valued metrics, set the plot type to **Heatmap chart**.

Monitoring metrics can be exported to BigQuery. Exporting metrics is necessary for performing in-depth analysis because Cloud Monitoring only saves data for a limited amount of time.

Here are a few more tips for tracking service account activity and keys that you should be aware of:

- Use **Activity Analyzer** to keep track of service accounts and service account keys
- Using the **Recommendation Hub**, you can see and manage all your project's proposals, including IAM recommendations
- If you have not used a service account in the last 90 days, **Service Account Insights** usage can help you locate it
- It is a good idea to disable accounts that have not been used in a while

So far, we have seen user-managed service accounts, but there are some service accounts that are created and managed by Google. We will take a brief look at them before moving on to IAM policy bindings.

Service agents

Google will create a service account for you in your project if you use one of the managed services (for example, Security Command Center or DLP). These are called **service agents**. Here are a couple of examples of what service agents look like:

- **Security Center Service Agent**: `service-53984177202@security-center-api-staging.iam.gserviceaccount.com`
- **Risk Manager Service Agent**: `organizations-720164443624@gcp-sa-riskmanager.iam.gserviceaccount.com`

The service agent typically has the following:

- The domain name used in the service agent's email address
- The role that the service agent is granted on the project
- After you activate an API that uses the service agent, Google grants the role automatically

There are some considerations when it comes to service agents:

- Service agent roles' permissions are subject to change at any time.
- Google can introduce new service agents at any time, both for existing services and for new services. In addition, the format of each service agent's email address is subject to change.
- Do not assign any principal impersonation ability to these service agents.

> **Note**
> Not all service agents can be impersonated but some allow impersonation.

- Keys for service agents cannot be created.

Now that we have covered service accounts in detail, we will move on and learn about IAM policy bindings in depth.

IAM policy bindings

In Google Cloud, access is managed through an IAM policy binding. An IAM policy is attached to a particular resource (remember, a resource could be a service account). An IAM policy contains a collection of role bindings that associate with one or more principals.

Recall that a principal could be one or more of the following:

- A Google account (a Gmail account is a form of Google account)
- A service account
- A Google group
- A Google Workspace account
- A Cloud Identity domain
- All authenticated users
- All users

IAM policy bindings (sometimes simply called bindings) associate a role to the principals both on the resource that the policy is attached to and on all the resource's descendants (for example, a policy attached to a folder is applicable to all the resources underneath that folder).

Policy structure

IAM policies consist of role definitions and additional details. A role binding determines the resources that can be accessed by specific users or groups. A role is associated with one or more principals, along with any applicable conditions. Metadata, including the etag and version, provides extra information about the policy.

Let us understand some guiding principles in creating IAM policies:

- There can be up to 1,500 principals in each IAM policy. Up to 250 unique groups can be specified in a single policy. Note that if you employ IAM conditions or grant roles to many principals with lengthy identifiers, then IAM may limit the policy to a smaller number of principals.
- IAM tracks the number of times each principal appears in the role bindings of the *allow policy*. No principals that exist in more than one role binding are deduplicated by this method. For example, the principal user `alice@acme.com` occurs in 50 role bindings in an allow policy, thus you could add 1,450 additional principals to the role bindings in the allow policy. Regardless of the number of individual members of a domain or Google group, the domain or group is counted as a single principal.
- Any additional information about the request, such as its origin or target resource, can be used as a condition to further restrict role binding. Conditional access is often used to determine whether a user's request can be granted. The term *conditional role binding* refers to a role binding that includes a condition. Some Google Cloud services do not accept conditions in IAM policies.

Now we understand the principles, let us look at the structure of the policy. The IAM policy includes the following fields:

- An `etag` field, which is used for concurrency control, ensures that policies are updated consistently. Concurrency control is supported by IAM via an `etag` field in the policy. Every time an IAM policy is updated, the value of the `etag` field is updated as well. As a result, an IAM policy with an `etag` field but no role bindings is ineffective.
- There is an optional `version` column that indicates the policy's schema version number.
- Additionally, the IAM policy can include an `auditConfig` field, which defines how each service logs activities for each service type, including organizations, folders, projects, and billing accounts.

Within 60 seconds of making an IAM policy modification, the new policy will take effect. The change can take up to seven minutes, however, to propagate across Google Cloud in some instances.

Here is an example of a simple policy:

```
{
  "bindings": [
    {
      "members": [
        "user:bob@acme.com"
      ],
      "role": "roles/owner"
    }
  ],
  "etag": " BwUjMhCsZpY =",
  "version": 1
}
```

The *Owner* basic role is provided to `bob@acme.com` without any restrictions in the simple policy example here. The access granted to `bob@acme.com` by this position is virtually limitless.

Here is an example of a policy that has multiple roles bound to it. Each role binding confers a distinct role to its recipient:

```
{
  "bindings": [
    {
      "members": [
        "user:bob@acme.com"
      ],
      "role": "roles/resourcemanager.organizationAdmin"
    },
```

```
    {
      "members": [
        "user:alice@acme.com",
        "user:bob@acme.com"
      ],
      "role": "roles/resourcemanager.projectCreator"
    }
  ],
  "etag": "BwUjMhCsZpY=",
  "version": 1
}
```

The Organization Admin predefined role (`roles/resourcemanager.organizationAdmin`) is granted to Jie (`jie@acme.com`) in the first role binding. Organizations, folders, and limited project operations are all included in this role's set of privileges. The Project Creator role (`roles/resourcemanager.projectCreator`) gives Jie (`jie@acme.com`) and Raha (`raha@acme.com`) the power to start new projects. Both Jie and Raha have access to fine-grained role bindings, but Jie has more access than Raha.

Example: Policy with conditional role binding

This example shows an IAM policy that binds principals to a predefined role and uses a condition expression to constrain the role binding:

```
{
  "bindings": [
    {
      "members": [
        "group:prod@acme.com",
        "serviceAccount:prod-example@acme-project-id.iam.
gserviceaccount.com"
      ],
      "role": "roles/appengine.deployer",
      "condition": {
          "title": "Expires_July_1_2022",
          "description": "Expires on July 1, 2022",
          "expression":
            "request.time < timestamp('2022-07-01T00:00:00.000Z')"
      }
    }
  ],
  "etag": " BwUjMhCsZpY =",
  "version": 3
}
```

The `version` field is set to `3` in this example because of a condition expression in the policy. The policy's role binding only lasts until July 1, 2022, and only for the Google groups `prod-dev@acme.com` and `prod-example@acme-project-id.iam.gserviceaccount.com`.

Let us see an example now that shows a policy with conditional and unconditional role bindings.

In this example, an IAM policy contains both conditional and unconditional role bindings for the same role:

```
{
"bindings": [
  {
    "members": [
      "serviceAccount:prod@acme-project-id.iam.gserviceaccount.com"
     ],
     "role": "roles/appengine.deployer"
  },
  {
    "members": [
      "group:prod@acme.com",
      "serviceAccount:prod@acme-project-id.iam.gserviceaccount.com"
    ],
    "role": "roles/appengine.deployer",
    "condition": {
      "title": "Expires_July_1_2024",
      "description": "Expires on July 1, 2024",
      "expression":
        "request.time < timestamp('2024-07-01T00:00:00.000Z')"
    }
  }
],
"etag": " BwUjMhCsZpY =",
"version": 3
}
```

The `serviceAccount:prod@acme-project-id.iam.gserviceaccount.com` service account is included in two role bindings for the same role in this case. No conditions are attached to the first binding. Until July 1, 2024, the second role binding grants the role.

In Google Cloud IAM, conditional role bindings take precedence over role bindings without conditions. When a conditional role binding is specified for a principal, it overrides any non-conditional role bindings for that principal. This means that the conditions defined in the conditional role binding will determine whether the principal is granted the specified role. If the conditions are not met, the non-conditional role bindings will not be considered for that principal.

The `group:prod@acme.com` Google group, on the other hand, is only included if the role binding conditions are met. As a result, it is only in effect until July 1, 2024.

Now let us look at how policy inheritance works in Google Cloud.

Policy inheritance and resource hierarchy

Policy inheritance describes how policies are applied to resources that are lower in the resource hierarchy. The term *effective policy* refers to a resource's policies being passed down through all its parents, all the way up the resource hierarchy. In that sequence, it is a combination of the following:

- The policy set on the resource
- The policies set on all of the resource's ancestry resource levels in the hierarchy

Each new role binding (acquired from parent resources) that has an impact on the resource's effective policy is assessed separately. If any of the higher-level role bindings provide access to the resource, a specific access request is granted. The access grant scope expands when a new role binding is added to any level of the resource's inherited policy.

To understand policy inheritance, let us consider a scenario where you grant a user, Bob, two different IAM roles at different levels in the resource hierarchy.

To grant Bob a role at the organization level, you set the following policy for your organization:

```
{
  "bindings": [
    {
      "members": [
        "user:bob@acme.com"
      ],
      "role": "roles/storage.objectViewer"
    }
  ],
  "etag": "BwUjMhCsNaY=",
  "version": 1
}
```

This policy grants Bob the Storage Object Viewer role (`roles/storage.objectViewer`). Since you set the policy at the organization level, Bob can use these permissions for all projects and all Cloud Storage objects in the organization.

To grant Bob a role at the project level, you set the following policy on the project:

```
{
  "bindings": [
    {
      "members": [
        "user:bob@acme.com"
      ],
      "role": "roles/storage.objectCreator"
    }
  ],
  "etag": " BwUjMhCsNaY =",
  "version": 1
}
```

This policy grants Bob the *Storage Object Creator* role (`roles/storage.objectCreator`), which lets him create Cloud Storage objects. Since this policy is set on the project level, Bob can create a cloud storage object only in the given project.

Let's summarize this particular example:

- A policy at the organization level grants the ability to list and get all Cloud Storage objects under this organization
- A policy at the project level, for the given project, grants the ability to create objects within that project

Now you have learned about how IAM policies work, let us look at some of the best practices.

IAM Conditions

You can provide access to principals only if certain requirements are met using IAM Conditions. A condition is part of the role binding. If a condition exists, access is only permitted if the condition expression evaluates to true. Each condition expression is made up of a series of logical statements that indicate which attributes should be checked. Let us take a look at the structure of IAM conditions.

The IAM policy binding structure looks like this. The `condition` object within that structure will have the conditions:

```
"bindings": [
  {
    "role": ...,
    "members": ...,
```

```
    "condition": ...
  },
  ...
]
```

The `condition` object has the following structure:

```
"condition": {
    "title": ...,
    "description": ...,
    "expression": ...
}
```

Let us go over the syntax of `condition`:

- `title` is required; however, `description` is optional.
- `expression` is required where you will define logic for conditional evaluation.
- You would use **Common Expression Language** (**CEL**) to define your logic using logical operators such as and (`&&`), or (`||`), or inverse (`!`) to define conditions. You can have up to 12 logical operators in each expression.

Check the Google Cloud documentation for a list of resources that accept condition role bindings and attributes supported by conditions.

Here are the different elements of conditions:

- **Variables**: These are attributes such as `request.time` (of type `Timestamp`) or `resource.name` (of type `String`). These variables are populated with values based on the context at runtime.
- **Operators**: Every data type (for example, `String` or `Timestamp`) supports a set of *operators* that can be used to create a logical expression, for example, `resource.service == "compute.googleapis.com"`. Operators are used for comparing values in a `resource.service` variable with a literal value. For example, `compute.googlepis.com`.
- **Functions**: These support more complex operations; for example, if you needed to match a part of a string, you would use `request.path.startsWith("/acmeBucket")`.
- **Logical operators**: Conditions support three logical operators: `&&`, `||`, and `!`. They make it possible to use multiple input variables in a condition expression. For example, `request.time.getFullYear() < 2020 && resource.service == "compute.googleapis.com"` joins two simple statements.

Policy best practices

The following best practices apply to organizations with many Google Cloud users:

- **Managing a large number of users**: When managing multiple user accounts with the same access, use Google groups. Each individual user account becomes a member of a group and the group is granted the required roles instead of individual user accounts.
- **Permissions granted at the organizational level**: Think about *which* principals have access permissions at the organizational level. Only a few *specific* teams (such as security and network teams) should have access to this level of the resource hierarchy in most organizations.
- **Grant permissions at the folder level**: When granting permissions at the folder level, consider using layers of folders to match your organization's operation structure, with each parent/child folder having various sets of access grants that are aligned with business and operation demands. A parent folder, for example, could represent a department, while one of its child folders could represent resource access and operation by a group, and yet another child folder could represent a small team. All folders may contain projects related to their team's operational requirements. This method of using folders ensures correct access isolation while adhering to policies inherited from the parent folder(s) and organization. When building and administering Google Cloud resources, this technique demands minimal policy maintenance.
- **Permissions provided at the project level**: When necessary, grant role bindings at the project level to adhere to the principle of least privilege. For example, if a principal requires access to three of the ten projects in a folder, you should grant access to each of the three projects separately; however, granting a role on the folder would grant the principal access to another seven projects that they do not require.
- **Granting permissions selectively**: You may also grant roles at the organization or folder level using IAM Conditions, but only for a subset of folders or projects.
- **Regularly review and audit IAM policies**: Conduct regular audits of IAM policies to identify any excessive or unnecessary permissions. Review the access granted to users, service accounts, and groups. Remove any outdated or unused permissions to minimize potential security risks.
- **Use IAM Conditions**: Leverage IAM Conditions to enforce additional security constraints based on contextual attributes such as IP address, time of day, or device status. IAM Conditions provides granular control over access to resources and can enhance security for specific use cases.
- **Implement monitoring and logging**: Enable monitoring and logging for IAM-related activities to detect and respond to any suspicious or anomalous behavior. Monitor and analyze logs to identify any unauthorized access attempts, policy violations, or unusual patterns of activity.
- **Educate and train users**: Provide regular training and awareness programs to educate users about IAM policies, best practices, and the importance of security. Promote a culture of security within the organization and encourage users to follow secure practices when managing IAM policies.

- **Use IAM Policy Intelligence tools**: As a part of the IAM Policy Intelligence portfolio, you have several tools to effectively manage permissions across your entire cloud organization. Use these tools effectively and train your developer community to utilize them.

In the next section, we will look at IAM Policy Intelligence briefly.

Policy Intelligence for better permission management

Policy Intelligence is a powerful feature of Google Cloud IAM that enables better permission management by providing insights, recommendations, and controls to optimize access control policies. With the ever-growing complexity of cloud environments and the need to balance security and productivity, Policy Intelligence offers valuable capabilities to enhance the overall governance and security posture of an organization. In this section, we will explore the various aspects of Policy Intelligence in Google Cloud IAM and how it can be leveraged for effective permission management. There are three main products in the Policy Intelligence suite:

- **IAM Recommender**: IAM Recommender is an AI-powered tool that helps optimize access control policies in Google Cloud. It provides recommendations for removing excessive or unnecessary permissions from IAM policies based on usage patterns, helping to ensure the principle of least privilege is followed.
- **Policy Analyzer**: Policy Analyzer is a feature within the Google Cloud console that analyzes IAM policies to identify potential security risks and violations. It scans policies for potential issues, such as overly permissive roles or resource-level permissions that can lead to security vulnerabilities.
- **Policy Troubleshooter**: Policy Troubleshooter is a feature of the Policy Intelligence suite that allows users to troubleshoot IAM permissions. It does this by comparing the requirements of resource permission to the permissions of a principal. This allows users to determine whether they have access to a resource based on their current IAM policy.

A critical aspect of Policy Intelligence is its ability to provide intelligent recommendations for permission changes. By analyzing historical data and access patterns, Policy Intelligence can identify over-permissive or under-permissive access policies and suggest modifications to align them with the principle of least privilege. These recommendations help organizations enhance their security posture by ensuring that access permissions are granted based on actual usage and business requirements. Implementing these recommendations can help reduce the attack surface, mitigate the risk of unauthorized access, and ensure that only the necessary permissions are granted to each user or service account.

Let us go over the best practices for permission management when using Policy Intelligence tools:

- **Analyze access patterns**: Utilize Policy Intelligence tools and features to analyze the collected access data. This analysis includes identifying access patterns, access frequencies, and potential security risks. By leveraging machine learning algorithms and advanced analytics, Policy Intelligence can provide valuable insights into access behaviors.

- **Review policy recommendations**: Policy Intelligence generates recommendations based on the analysis of access patterns and behaviors. Review these recommendations to identify areas where permissions can be optimized. The recommendations may include suggestions to remove excessive privileges, modify existing roles, or create new custom roles.
- **Implement recommended changes**: Apply the recommended changes to your access control policies. This may involve modifying existing roles, creating custom roles, or adjusting permission assignments for specific identities or groups. Follow the principle of least privilege to ensure that users and service accounts have only the necessary permissions required to perform their tasks.
- **Combine with other security best practices**: Remember that Policy Intelligence is a complementary feature within the Google Cloud IAM ecosystem. It should be used in conjunction with other security best practices, such as strong authentication mechanisms (for example, multi-factor authentication), regular security audits, and adherence to security frameworks and compliance standards.

By following these steps, organizations can effectively leverage Policy Intelligence in Google Cloud IAM for better permission management. This approach enables proactive identification of security risks, optimization of access control policies, and continuous monitoring of policy compliance, enhancing the overall security and governance of your cloud resources.

So far, we have looked at IAM policy best practices. Now let us briefly go over IAM tags and how they can be used for access control.

Tag-based access control

Tags are key-value pairs that can be attached to organizations, folders, or projects. They are an IAM construct and differ from labels and network tags. Tags follow an inheritance model, where a tag applied at the organization level is inherited by child objects, but this inheritance can be overridden if needed. Conditional IAM roles can be granted based on specific tags assigned to a resource.

In the resource hierarchy, tags are automatically inherited, but you can attach an additional tag to a resource to prevent it from inheriting a specific tag value. Essentially, each tag on an organization or folder sets a default value, which can be overridden by tags on lower-level resources such as folders or projects. Once tags are attached to a resource, you can define conditions to grant access based on those tags.

Tag structure

Here are how tags are structured in IAM:

- A *tag* is a key and value pair.
- A *permanent ID*, which is globally unique and can never be reused.

- A *short name* for each key must be unique within your organization, and the short name for each value must be unique for its associated key. For example, a tag key could have the short name `env`, and a tag value could have the short name `prod`.
- A *namespaced name*, which adds an organization ID to the short name of a tag key. For example, a tag key could have the namespaced name `123456789012/env`.

Best practices for tags

The following are some of the best practices when using tags:

- **Consistent tagging strategy**: Establish a consistent tagging strategy across your organization. Define clear guidelines for how tags should be structured and used. Consistency ensures that tags are meaningful, standardized, and easily understood by administrators and users.
- **Tag resource hierarchy**: Align your tag structure with your resource hierarchy. Consider organizing tags in a way that mirrors your project, folder, or organization structure. This helps in efficient management and makes it easier to associate tags with specific resources.
- **Use descriptive tags**: Use descriptive tags that accurately describe the purpose, ownership, or attributes of a resource. This helps in quickly identifying and categorizing resources, especially when dealing with a large number of assets.
- **Security and compliance tags**: Implement security- and compliance-related tags to identify resources that have specific security requirements or compliance obligations. These tags can help enforce access controls, ensure proper encryption, or track resources subject to specific regulations.
- **Tag resource owners**: Include tags to identify resource owners or responsible individuals or teams. This allows for better accountability and facilitates communication among stakeholders. It can be useful in managing permissions, delegating responsibilities, and streamlining resource management processes.
- **Automation and tagging policies**: Leverage automation tools and scripting capabilities to enforce tagging policies. Automate the application of specific tags based on predefined rules or criteria. This helps ensure consistency, reduces manual errors, and streamlines the process of managing tags across a large number of resources.
- **Regular tag reviews**: Conduct regular reviews of tags applied to resources. Remove unused or outdated tags and ensure that tags remain accurate and up to date. Regular reviews help maintain data integrity, optimize resource management, and avoid potential confusion or misinterpretation of tag meanings.
- **Tag-based IAM policies**: Utilize tag-based IAM policies to simplify access management. Assign IAM roles and permissions based on tags rather than individual resources. This allows for easier administration, especially when resources are dynamically created or changed frequently.

- **Monitor and audit tags**: Implement monitoring and auditing mechanisms to track changes to tags and associated resources. Monitor tag usage, modifications, and policy enforcement to ensure compliance and identify any unauthorized changes or deviations.
- **Education and training**: Educate and train your teams on the importance of using tags effectively. Provide guidelines and documentation to ensure that everyone understands the purpose and significance of tagging. Promote a culture of tag awareness and adherence to best practices.

So far, we have looked at IAM; now we will switch gears and look at access control for Cloud Storage. Cloud Storage is one of the original Google Cloud products and has evolved to have some fine-grained access policies (ACLs), so we will also look at those.

Cloud Storage ACLs

Cloud Storage provides separate access control in addition to IAM. There are two ways, in fact: a more uniform way of doing access control via IAM and a legacy way of doing access control via fine-grained ACLs. Object ACLs do not appear in the hierarchy of IAM policies, so be aware of how your Cloud Storage buckets are controlled. When evaluating who has access to one of your objects, make sure you check the ACLs for the object, in addition to checking your project- and bucket-level IAM policies. This could get very convoluted, so the recommendation is to use uniform access control using IAM in most cases.

Access Control Lists (ACLs)

You can use an ACL to determine who has access to your buckets and objects, as well as what level of access they have. ACLs are applied to specific buckets and objects in Cloud Storage. There are one or more entries in each ACL. An entry enables a certain user (or group) to carry out specific tasks. Each entry is made up of two bits of data:

- A *permission*, which defines *what* actions can be performed (for example, read or write)
- A *scope* (sometimes referred to as a *grantee*), which defines *who* can perform the specified actions (for example, a specific user or group of users)

Consider the following scenario: you have a bucket from which you want anonymous users to be able to access objects, but you also want your collaborator to be able to add and delete objects from the bucket. Your ACL would be made up of two entries in this case:

- In one entry, you would give `READER` permission to a scope of `allUsers`
- In the other entry, you would give `WRITER` permission to the scope of your *collaborator* (there are several ways to specify this person, such as by their email)

Here are some points to consider when using ACLs:

- You can only make up to 100 ACLs for a bucket or object.
- When the entry scope is a group or domain, it counts as one ACL entry, no matter how many users are in the group or domain.
- When a user tries to access a bucket or storage object, the Cloud Storage system looks at the bucket or object's ACL. If the ACL gives the user permission to do what is being asked, the request is granted. If not, the request fails, and a **403 Forbidden** error is sent back.

Let us look at ACL permissions now.

ACL permissions

Permissions describe *what* can be done to a given object or bucket. Cloud Storage lets you assign concentric permissions for your buckets and objects within the bucket.

What are concentric permissions?

You do not have to assign multiple scopes to ACLs in Cloud Storage to give more than one permission. When you give someone `WRITER` permission, you also give them `READER` permission, and when you give someone `OWNER` permission, you also give them `READER` and `WRITER` permission.

Based on how objects are uploaded, object ACLs are applied accordingly:

- **Authenticated uploads**: You (the person who uploaded the object) are listed as the object owner. Object ownership cannot be changed by modifying ACLs. You can change object ownership only by replacing an object. The project owners and project editor group have `OWNER` permission on the object.
- **Anonymous uploads**: If an anonymous user uploads an object, which is possible if a bucket gives the `allUsers` group `WRITER` or `OWNER` permission, the object gets the default bucket ACLs.

Best practices for Cloud Storage access while using ACLs

ACLs require active management, like other administrative settings. Before sharing a bucket or object with other users, consider who you want to share it with and what function each person will play. Changes in projects, usage patterns, and organizational ownership may need you to modify bucket and object ACL settings over time, especially if you manage them for a large organization or a large team. Consider these recommended practices when evaluating and planning your access control settings:

- Do not use sensitive information in the Cloud Storage bucket name.
- Follow the principle of least privilege when granting access to buckets and objects. This will make sure that data contained in the bucket is not accidentally exposed.

- Do not grant `OWNER` permission to anyone in the organization. Typically, you want the IaC service account to have ownership of the buckets.
- Be aware of how you grant permissions to anonymous users. The `allUsers` and `allAuthenticatedUsers` scopes provide access to everyone on the internet – be they anonymous users or users logged in with a Google account. You may not want this behavior.
- Avoid setting ACLs that result in inaccessible objects. An inaccessible object is an object that cannot be downloaded (read) and can only be deleted. This can happen when the owner of an object leaves a project without granting anyone else `OWNER` or `READER` permission on the object.
- Make sure to delegate administrative control of your buckets. By default, *project owners* have `OWNER` permissions. Make sure to not assign this role to individuals.
- Make sure you understand Cloud Storage's interoperable behavior before using it.

Uniform bucket-level access

Cloud Storage has uniform bucket-level access to support the system's uniform permissions. When you use this feature on a bucket, all ACLs for Cloud Storage resources in the bucket are disabled, and access to Cloud Storage resources is allowed solely through IAM. You have 90 days after enabling consistent bucket-level access to revoke your decision.

Please keep in mind that enabling uniform bucket-level access revokes access for users who get access purely through object ACLs and prevents users from managing rights through bucket ACLs. Before enabling consistent bucket-level access, read the Google Cloud product documentation when moving an existing bucket and consider whether it's the right thing to do in your situation.

Using uniform bucket-level access in practice

When should you use uniform bucket-level access? Uniform bucket-level access streamlines how you offer access to Cloud Storage resources. Using consistent bucket-level access also enables you to use features such as Domain Restricted Sharing and IAM Conditions.

You might not want to use uniform bucket-level access and instead keep fine-grained ACLs in the following situations:

- You want legacy ACLs to control bucket object access
- You want an object's uploader to have full control over that object but others to have less access to the objects
- You want to use the XML API to view or set bucket permissions

IAM Conditions can only be used on buckets with uniform bucket-level access to prevent conflicts between policies and ACLs. So, note the following points when using it:

- Set IAM conditions on a bucket after enabling uniform bucket-level access
- Remove all IAM conditions from a bucket's policy to deactivate uniform bucket-level access
- Uniform bucket-level access cannot be disabled after 90 days of use

This concludes the section on IAM policies and storage bucket ACLs. Let us look at IAM APIs.

IAM APIs

IAM uses the following API endpoints (regular OAuth access tokens either for a user or service account can be used to access these APIs):

- Policies (v2)
- Roles (query and get/list)
- Organizations roles
- Permissions
- Projects
    - IAM policies (linting and querying)
    - Workload identity pools, operations, and providers
    - Permissions
    - Roles
    - Service accounts
    - Service account keys
- Service account credentials
- Security token services

Finally, let us look at various log files for IAM APIs. You will often start with these logs to troubleshoot an access issue.

IAM logging

Google Cloud IAM writes audit logs and admin logs to help with questions such as "*Who did what, where, and when?*" These logs are vitally important for audit and forensic capabilities.

For information on Admin Activity and Data Access read audit logs, please check the Google Cloud product documentation.

IAM audit logs use one of the following resource types:

- `api`: A request to list information about multiple IAM roles or policies
- `audited_resource`: A request to exchange credentials for a Google access token
- `iam_role`: An IAM custom role
- `service_account`: An IAM service account, or a service account key

Log name

Let us assume `project_id` = `acme-project-id`, `folder_id` = `acme-folder`, `billing_account_id` = `123456`, and `organization_id` = `987654321`:

```
projects/acme-project-id/logs/cloudaudit.googleapis.com%2Factivity
projects/acme-project-id/logs/cloudaudit.googleapis.com%2Fdata_access
projects/ acme-project-id/logs /cloudaudit.googleapis.com%2Fsystem_event
projects/acme-project-id/logs/cloudaudit.googleapis.com%2Fpolicy

folders/acme-folder/logs/cloudaudit.googleapis.com%2Factivity
folders/acme-folder/logs/cloudaudit.googleapis.com%2Fdata_access
folders/acme-folder/logs/cloudaudit.googleapis.com%2Fsystem_event
folders/acme-folder/logs/cloudaudit.googleapis.com%2Fpolicy

billingAccounts/987654321/logs/cloudaudit.googleapis.com%2Factivity
billingAccounts/987654321/logs/cloudaudit.googleapis.com%2Fdata_access
billingAccounts/987654321/logs/cloudaudit.googleapis.com%2Fsystem_event
billingAccounts/987654321/logs/cloudaudit.googleapis.com%2Fpolicy

organizations/987654321/logs/cloudaudit.googleapis.com%2Factivity
organizations/987654321/logs/cloudaudit.googleapis.com%2Fdata_access
organizations/987654321/logs/cloudaudit.googleapis.com%2Fsystem_event
organizations/987654321/logs/cloudaudit.googleapis.com%2Fpolicy
```

These are the logs that you should be familiar with for Cloud IAM. You should also understand service account-specific activities that get logged into these logs. Let us look at service account-specific information now.

Service account logs

One of the critical aspects of working with Google Cloud is to be able to identify service accounts' activities. The following service account activities are logged by the IAM API:

Note

For details on the relevant fields of log entries, please look at the Google Cloud product documentation. You might see log entries in which the `service-agent-manager@system.gserviceaccount.com` service account grants roles to other Google-managed service accounts. This behavior is normal and expected.

- Creating service accounts
- Granting a service account a user role
- Granting access to a service account on a resource
- Setting up a Compute Engine instance to run as a service account
- Accessing Google Cloud with a service account key
- Creating a service account key
- Authenticating with a service account key
- Impersonating a service account to access Google Cloud
- Authenticating with a short-lived credential

This concludes various aspects of IAM logging. It's important to understand this very well as you might have to develop security alerts based on your security operations requirements.

Summary

In this chapter, we explored various powerful features of Cloud IAM, including principals, roles, IAM policies, and service accounts. We gained insights into effective service account key management and learned how to detect potential issues when keys are checked into Git. Additionally, we discovered the versatility of IAM conditions and adopted best practices for creating robust IAM policies. We also delved into Cloud Storage ACLs and their ability to provide fine-grained access control. Armed with this knowledge, you are now equipped to confidently set up IAM policies for any workloads in Google Cloud, troubleshoot access problems, and implement the recommended best practices we discussed. We even delved into advanced features such as IAM Policy Intelligence and WIF.

As we conclude this chapter on Google Cloud's IAM features, the upcoming chapters will focus on exploring the robust network security capabilities of Google Cloud.

Further reading

- Attribute reference for IAM Conditions: `https://packt.link/wMFb4`
- Predefined ACLs for Cloud Storage: `https://packt.link/jfhAt`
- IAM Relation to Cloud Storage ACLs: `https://packt.link/gQoHG`
- Cloud Storage ACL behavior: `https://packt.link/VgFY0`
- Cloud Storage interoperability with AWS S3: `https://packt.link/vqNy3`
- Tags using conditions: `https://packt.link/zzvld`

7
Virtual Private Cloud

In this chapter, we will look at Google Cloud **Virtual Private Cloud** (**VPC**). The chapter will focus on the network security concepts within Google Cloud, although we will cover some foundational networking concepts that are essential. We will begin by looking at how Google Cloud's global footprint is structured with regard to different regions and zones. Then, we will cover the key concepts of VPC, such as global VPC versus regional, how to create custom IP address subnets, micro-segmentation, custom routing, and the different types of firewall rules and their usage. We will look at some design patterns for VPC, focusing on shared VPC in particular and its importance from a security perspective, and we'll get an overview of the different types of connectivity options that are available to connect Google Cloud to your on-premises or third-party cloud provider using different options. Finally, we will look at DNSSEC and some load balancer options available to handle TLS.

In this chapter, we will cover the following topics:

- Introduction to VPC
- Google regions and zones
- VPC deployment models
- Micro-segmentation
- Cloud DNS
- Load balancers
- Hybrid connectivity options
- Best practices and design considerations

Overview of VPC

Google Cloud VPC is a virtual network built on Google's internal production network and is based on Andromeda, which is Google's implementation of its software-defined network. Andromeda is out of scope for the exam, but those who want to understand more about the underlying network and how it is built can read more about it by using the link in the *Further reading* section.

You can think of Google Cloud VPC as a virtual network, which is very similar to a physical environment but more powerful due to its software-defined nature. At a high level, VPC is responsible for providing underlying connectivity to your Google Cloud services, such as **Compute Engine** (**virtual machines**), **Google Kubernetes Engine** (**GKE**), **Google App Engine** (**GAE**), and any other Google Cloud services that are built on top of these services. VPC also natively provides load balancing and proxy services, helps distribute traffic to backend services, and provides the capability to connect your Google Cloud to either your on-premises data center or other cloud service providers.

Let's understand some of the key features that VPC offers, as it will help you understand what capabilities are offered and how to use them in your network design:

- A VPC network is a global resource, including its components, such as the routes and the firewall rules. We will look at global/regional/zonal resources in the *Google Cloud regions and zones* section.
- With VPC, you have the ability to create IPv4-based subnets. These subnets are regional resources. IPv6 is also supported, where you can enable IPv6 on supported subnets; the only exception is that an auto-mode VPC network cannot have IPv6, but if you add a custom subnet to the auto mode VPC, you can enable IPv6 on that subnet. The important thing to note here is that auto mode VPC networks do not support dual-stack subnets.
- Firewall rules are used to control the traffic from the instances. When you create firewall rules for the network, although they control the network traffic, these rules are in fact applied to the VM interface. This then lets you control which VMs can communicate with each other on which ports. We will cover firewall rules in more detail in the *Micro-segmentation* section of this chapter.
- When you create a VPC network, all the resources inside the network with IPv4 addressing can communicate with each other without the need for any additional rules. Some restrictions may apply if you create firewall rules to control the traffic inside the VPC network.
- VM instances that are created inside the VPC network and have a private Ipv4 address can communicate with Google APIs and services. Google Cloud has private access, which we will cover later in this chapter, that lets instances communicate with Google APIs and services even if they don't have an external IP address.
- Fine-grained identity and access management policies are available for network administration. You can create custom roles to give the relevant privileges to your network team to manage VPC and its associated components.
- Shared VPC is a deployment model that can be used to centralize network functions in a common host project. Other services' projects can consume network services from the shared VPC host project, such as IP address, subnet, and hybrid connectivity, as long as they are part of the same organization.

- VPC peering is a concept used to connect a VPC network in one project to another or across two different organizations. Permissions are required at each end to authorize the creation of a VPC peer with no IP address overlaps.
- There are multiple options to create secure hybrid connectivity; you can connect your on-premises environment to Google Cloud using a VPN or a dedicated interconnect. We will look at hybrid connectivity in more detail later in the chapter, under the *Hybrid connectivity options* section.
- For use cases where you need to access Secure Access Service Edge and **Software Defined-Wide Area Network** (**SD-WAN**) and you require the use of **Generic Routing Encapsulation** (**GRE**), you can use VPC, which supports GRE. Other options such as VPN and Interconnect support the same capability. You can terminate GRE on your instance and forward the traffic to its respective location.

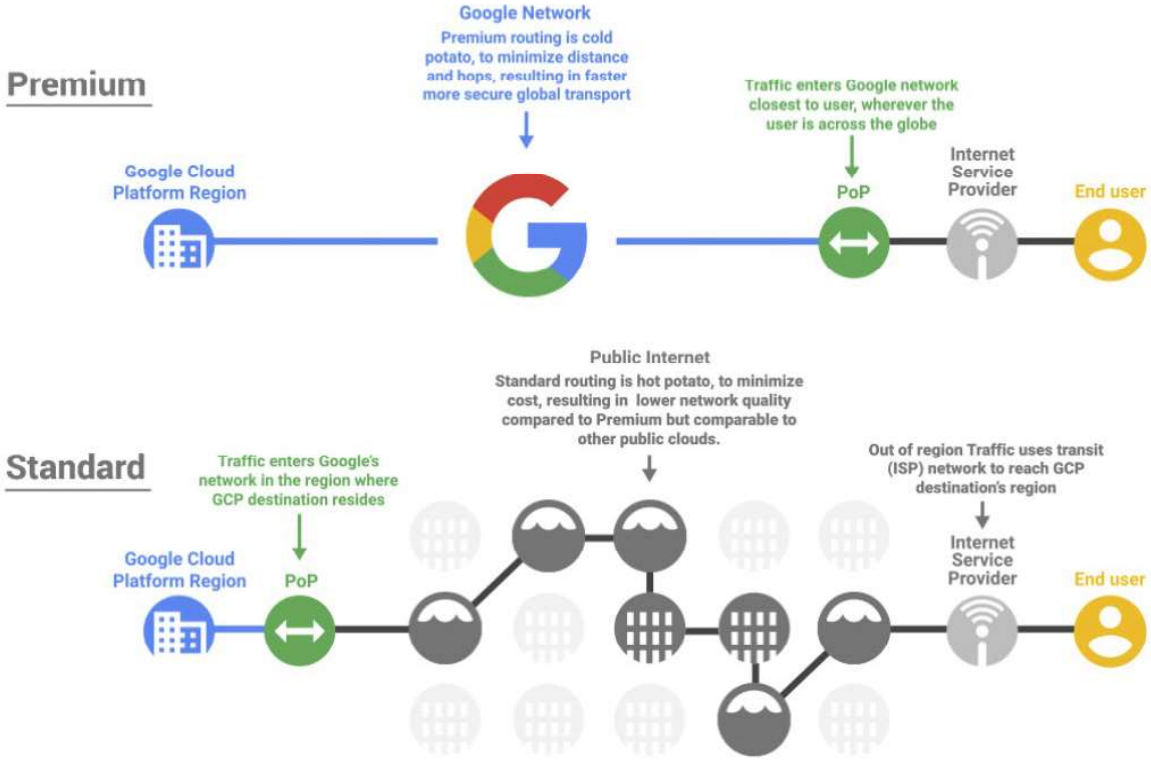

Figure 7.1 – Google Cloud network tiers

Figure 7.1 shows the network tiers that Google provides. Google offers two tiers, Standard Tier and Premium Tier, that can be configured at the VPC level. In a Premium Tier network, Google utilizes its own private network to route the traffic. From an end user perspective, if the user is trying to access services hosted on Google Cloud, the traffic will connect to the closest Google **point of presence** (**PoP**) location. Google has hundreds of PoP locations across the globe. From the PoP location, the

traffic will not traverse the internet and will instead use Google's own operated and managed fully encrypted private network. This provides better reliability and security.

The Standard Tier option is what most service providers use, that is, *hot potato*, meaning when a user tries to access the services hosted on Google Cloud, the service will go via the internet, which is non-deterministic and will not use Google's private network. This, although a more cost-effective option, does not provide the reliability, security, and availability of a Premium Tier network.

In the *VPC deployment models* section, where we will walk through the setup of VPC, we will highlight where you can configure the network tiers.

Google Cloud regions and zones

This topic is not part of the exam blueprint. Due to the fact that this is related to the design and architecture and how services are organized, it's important to understand some of the foundational concepts of how Google organizes its regions and zones. We will also look at what global, regional, and zonal resources are.

Regions refer to geographic locations where Google Cloud has data centers that offer cloud services. For example, in the United States, Google has regions in Iowa (us-central), Virginia (us-east), and Los Angeles (us-west); across the globe, Google Cloud has regions in Australia, Singapore, India, the UK, Germany, and more. A full list of Google Cloud regions can be found here: `https://packt.link/b9TVP`.

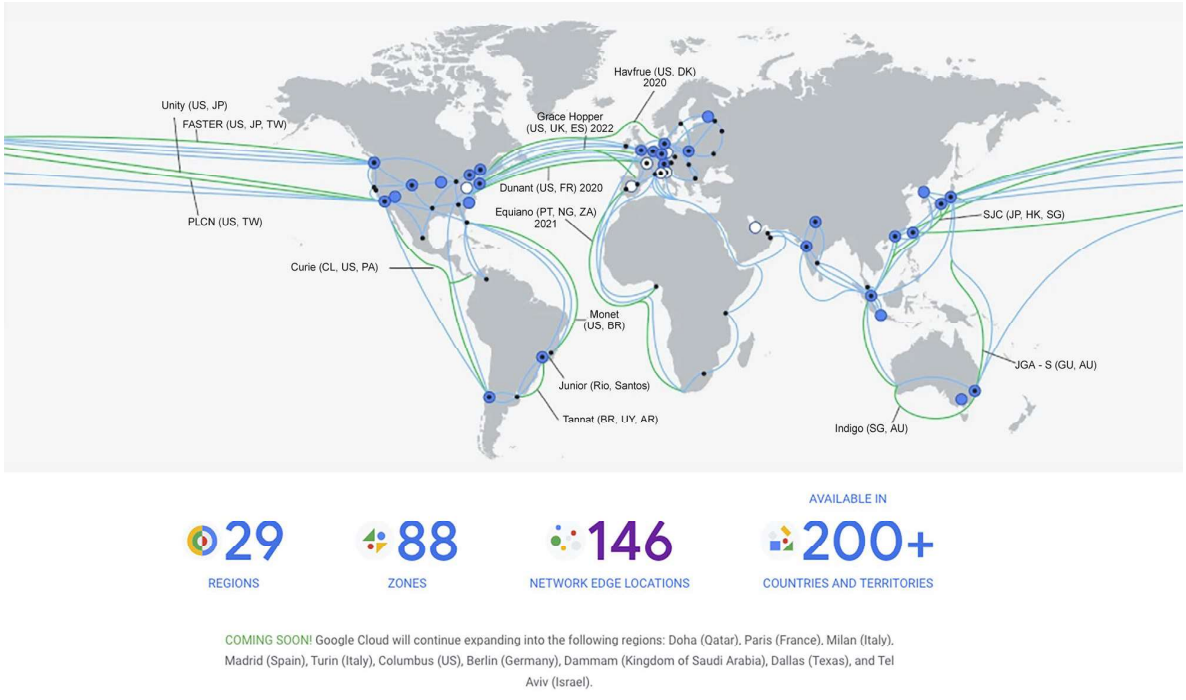

Figure 7.2 – Google Cloud region locations

At the time of writing, Google has a presence in more than 200 countries, with 29 regions, 88 zones, and 146 PoP or edge locations. As part of Google Cloud's strategy to expand, there are more regions announced every year.

Each region has at least two zones, and some have more than two zones to provide additional capacity and resilience. This is dictated by demand. As a rule of thumb, there are always two zones to provide fault tolerance and high availability. In order to address the resiliency and availability of your services, it is highly recommended to use multiple zones and regions. Both regions and zones have a standard naming convention to identify them, which is `<region>-<zone>`, for example, `us-central1`. Furthermore, `a`, `b`, or `c` is used to identify the zone, for example, `us-east1-a`. You can select any of these zones to deploy your cloud services, such as compute instances.

Google Cloud service availability is also region-specific. You can see whether the service you want is available in the region of your choice by navigating to `https://packt.link/ehHg8`.

We will now take a look at different types of resources that are either global, regional, or zonal in nature.

As the name suggests, global resources are global in nature, which means when configured inside a project, these resources can be accessed by any other resource in any zone as long as they are in the same project. VPC networks, firewall rules, routes, images, and snapshots are all examples of global resources.

Regional resources are only available in the region; for example, the subnets that you create are accessible only in the regions they are in and cannot cross a regional boundary. Some examples of regional resources include regional managed instance groups and cloud interconnects.

Zonal or per-zone resources are created in a zone and are available only within that zone. A compute instance that is created requires a zone to be specified where the resource will exist. This resource can access other global or regional resources but cannot access any other per-zone resource, such as a disk.

> **Note**
>
> For more details on which services are global, regional, or zonal, you can refer to `https://packt.link/4lg8M`.

VPC deployment models

VPC can be deployed in different models based on your network architecture requirements. In this section, we will look at the different models that are available and how you can create them. We will look at the two VPC modes that can be used to create a VPC network: auto mode and custom mode. After that, we will look at Shared VPC networks, what they are, and how to create one. In the next section, will go over VPC peering, a technique for connecting two VPCs that are located in different organizations but can be considered peers.

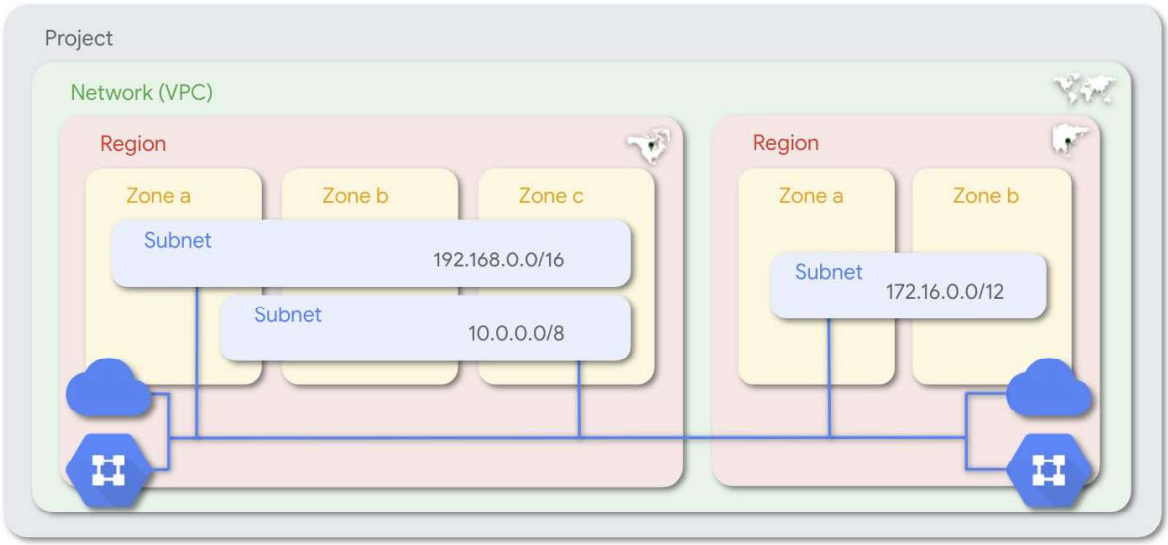

Figure 7.3 – Google network components

We will use the illustration shown in *Figure 7.3* to understand a few concepts before we jump into the deployment models. A VPC as a resource is created inside the project and is global in nature. Once you create the VPC network, you have the option to either create a global VPC with all regions in scope or customize things based on which region(s) you want to use. We will look at this in the next section on how to create a VPC using the two modes. Within each region, there are zones. You can select one, two, or three (if available) zones to create resources. For example, to create a compute instance, you need to specify which zone you want to use in which region. Furthermore, you can create subnets inside the zone. We will cover how to create subnets in the following sections, but in Google Cloud, subnets are regional resources; therefore, they can span across zones, as shown in *Figure 7.3*. In summary, you can create a single VPC network spanning multiple regions and provide connectivity to your cloud resources.

VPC modes

A VPC can be deployed in two different modes, as dictated by the subnet creation process.

Creating an auto mode VPC network

The first mode is auto mode, where the default network is created and one subnet per region is added automatically. In auto mode, all IP address ranges are pre-defined to `10.128.0.0/9`, and new subnets are also automatically added when new regions are made available. As the name suggests, auto mode essentially creates a global network in a project. You do have the option to create custom subnets for an auto mode VPC and, if required, you can also convert an auto mode VPC network to a custom VPC network as a one-way conversion, but you cannot convert a custom mode VPC network to auto mode. Auto mode VPC network creation is enabled by default when you create a project; if you want to disable the default behavior, you can do so by creating an organizational constraint policy called `compute.skipDefaultNetworkCreation`.

Let's take a look at how you can create an auto mode VPC network on Google Cloud:

1. Navigate to the **VPC networks** section from the Google Cloud menu: `https://packt.link/dYLnN`.
2. Next, click on **Create a VPC network**.
3. Fill in **Name**, following the naming convention.
4. Optionally, fill in **Description** as well for the network.
5. Next, under the **Subnets** section, select **Automatic** under **Subnet creation mode**.

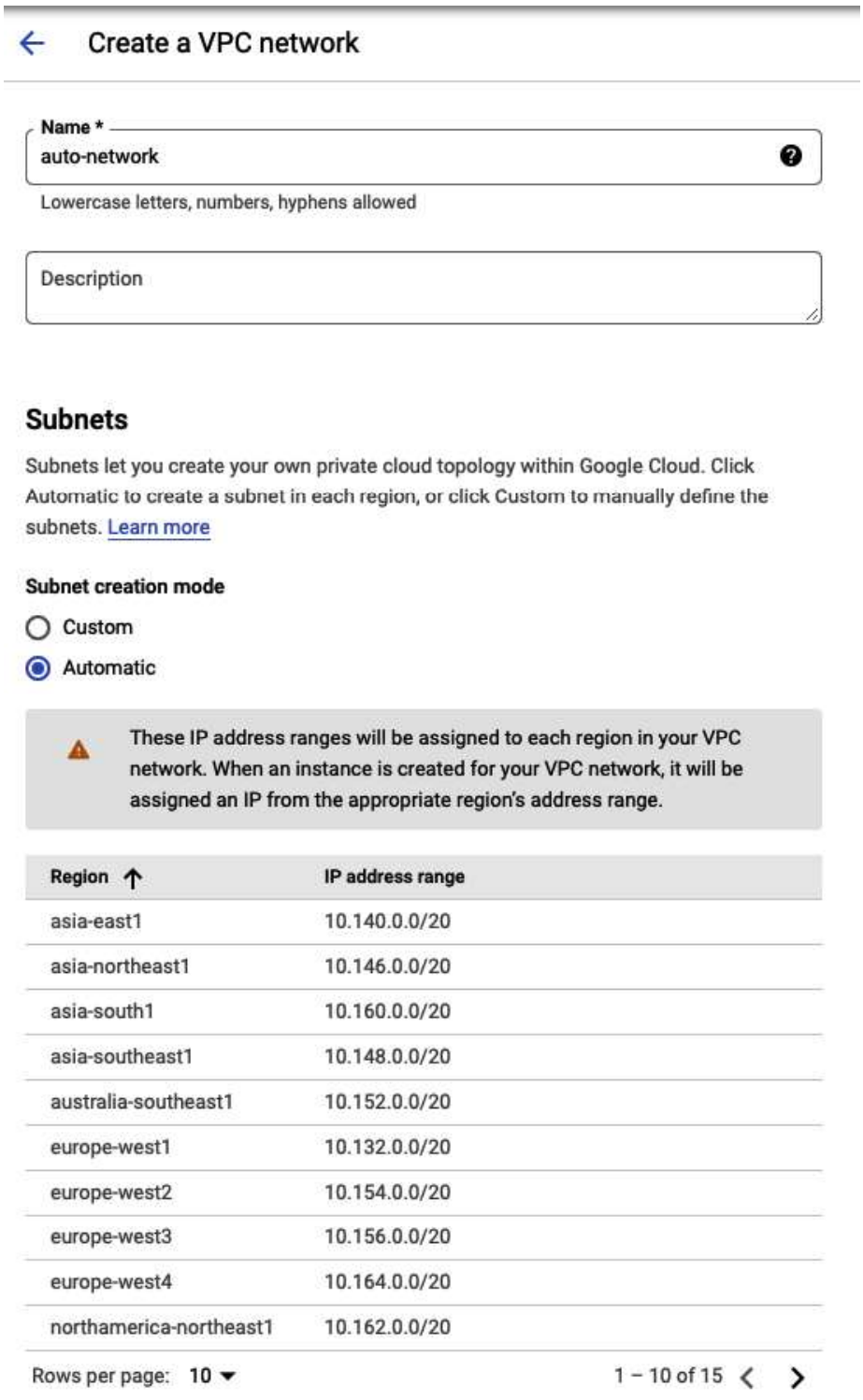

Figure 7.4 – VPC creation – auto mode

6. You can remove the default firewall rules established depending on the use case. There are two rules that prohibit all incoming traffic but permit all outgoing traffic; these rules cannot be deleted.

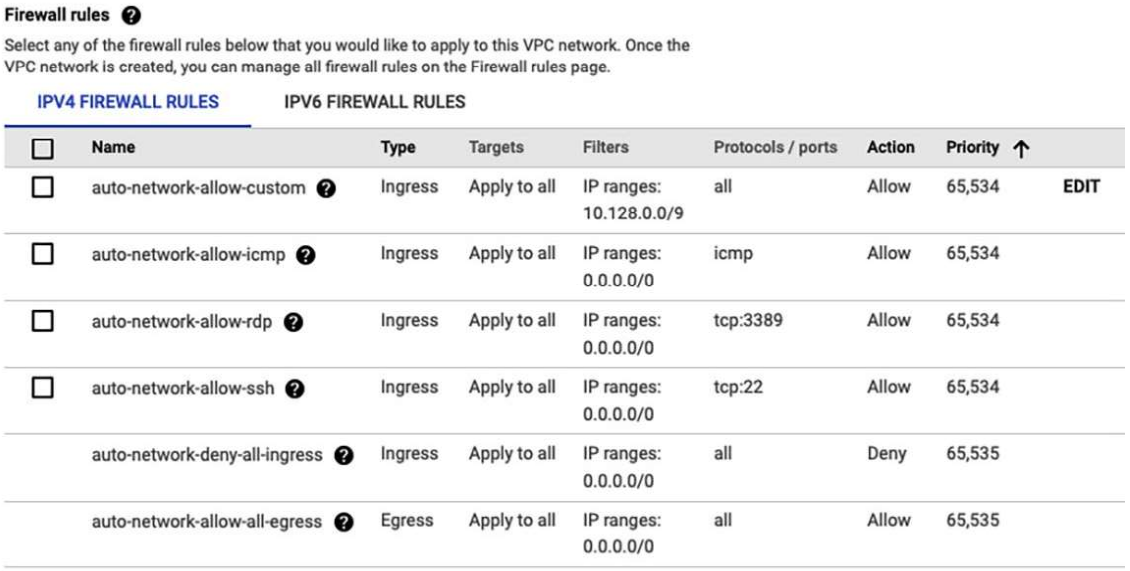

Figure 7.5 – Default firewall rules

7. Under **Dynamic routing mode**, select either **Regional** or **Global** based on your network requirements.
8. Leave the **Maximum Transmission Unit** (**MTU**) settings at their defaults, unless you need to change them.
9. Click **CREATE** for an auto mode VPC network to be created.

Creating a custom mode VPC network

Custom mode VPC creation gives you more control to select the regions where you want to have the network created and the flexibility to specify your own custom RFC1918-based IP address range for the subnets. For use cases where you need to create VPC peering, you need to create a custom network, as an auto mode network creates the same IP address range in every project. Therefore, to avoid IP address overlap, you need to be in control of the IP ranges when creating peering and hybrid connectivity with your on-premises network.

The steps to create a custom VPC network are as follows:

1. Navigate to the **VPC networks** section from the Google Cloud menu: `https://packt.link/ek0Sw`.
2. Next, click on **Create a VPC network**.
3. Provide a name following the naming convention.
4. Optionally, provide a description of the network.
5. Next, set the subnet creation mode to **Custom**.
6. Next, click on **New subnet** and provide the following information:

    I. The name of the subnet.

    II. An optional description of the subnet.

    III. Select the region you want to use.

    IV. Specify the RFC1918 subnet range. Refer to this link for the ranges that can be used: `https://packt.link/yko0P`.

    V. Create a secondary IP subnet range, if required. Secondary IP address ranges are used for allocating different IPs to multiple microservices running in a VM (for example, containers and GKE pods). You can change and extend a primary IP range but not a secondary IP range.

    VI. Specify whether you want to use **Private Google Access**. If you're not sure, you can always edit this and update it later.

    VII. Select whether you would like to turn the flow logs on or off. This attribute can again be changed by editing.

    VIII. Click on **DONE** when finished.

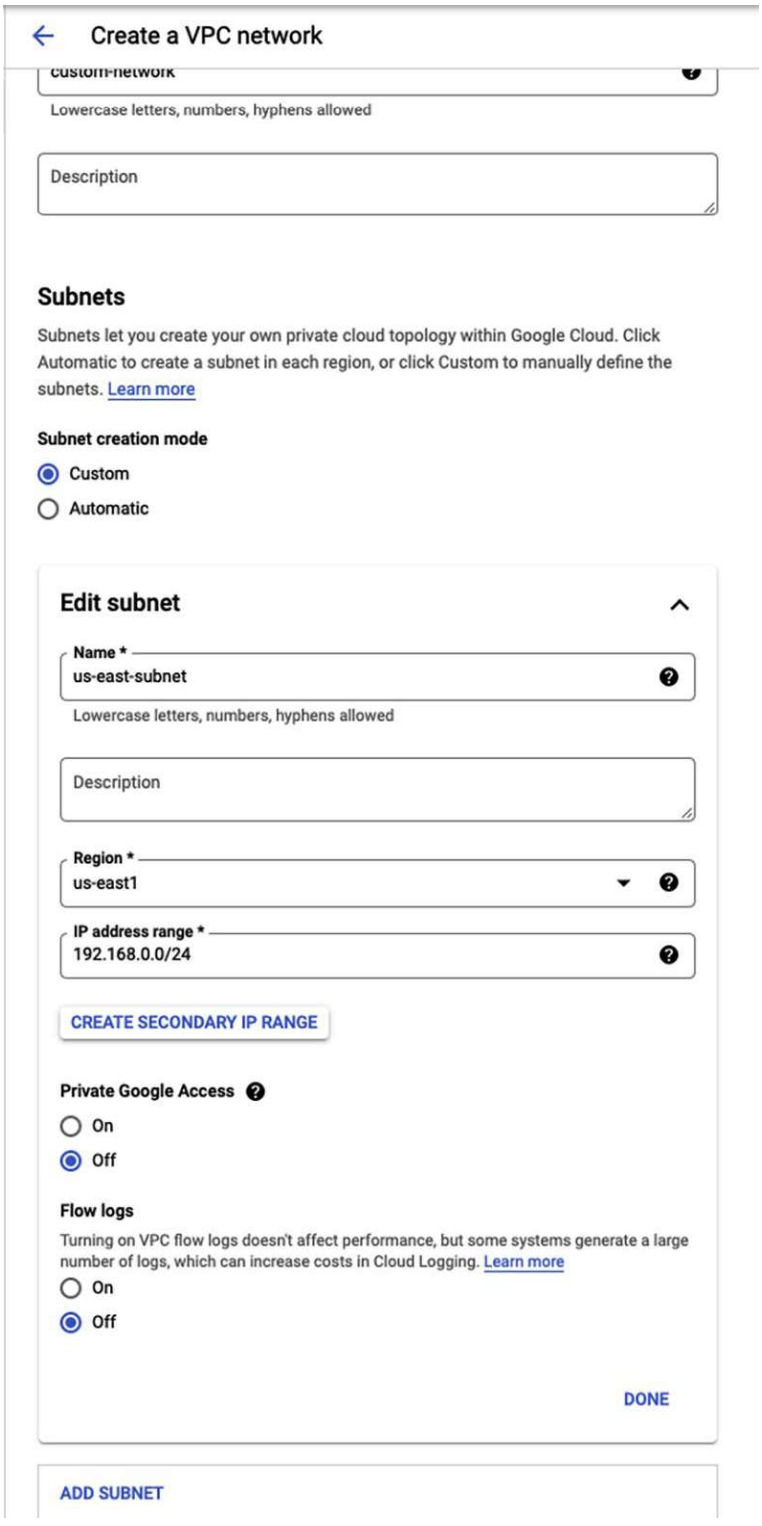

Figure 7.6 – Custom VPC creation

7. To add more subnets, you can repeat the preceding steps.
8. In the same way that auto mode default firewall rules are added based on typical use cases, you can also remove them. However, you cannot get rid of the two rules that block all incoming traffic and let all outgoing traffic through.
9. Under **Dynamic routing mode**, select either **Regional** or **Global** based on your network requirements.
10. Leave the **Maximum transmission unit** (**MTU**) settings at their defaults, unless you need to change them.
11. Click **CREATE** for a custom mode VPC network to be created.

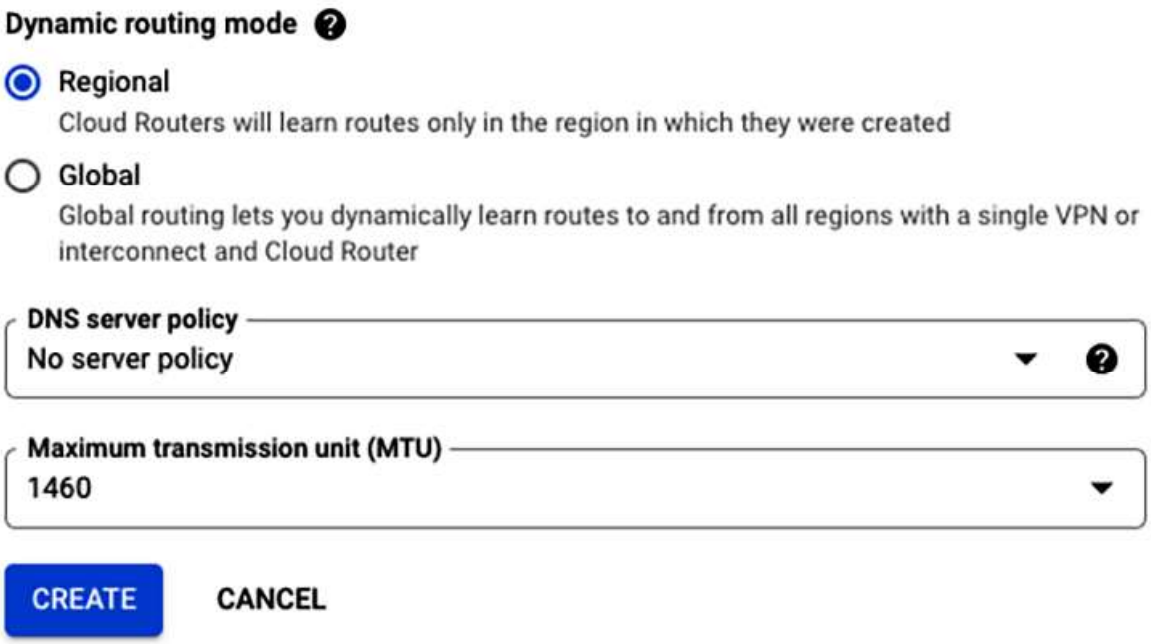

Figure 7.7 – Dynamic routing

This concludes the creation of a VPC network. Next, we will look at what a Shared VPC network is and how to create one using the Google Cloud console.

Shared VPC

A Shared VPC network is like any other VPC – the difference is that it is shared. You create a Shared VPC network inside a **host project**. Network administrators can control via a single pane of glass all the routes, subnets, firewall rules, interconnect/VPN, logs, and more inside the Shared VPC network. You can also create an organizational policy to restrict Shared VPC networks to host projects. The point of a Shared VPC network is to centrally control and manage all the network services that are offered by VPC. **Service projects** can attach to a host project, and services such as VMs and the resources

provisioned on them can allocate an IP from the Shared VPC network. Billing for resources that use the Shared VPC network is attributed to the service project. Permissions can be granted at the VPC or subnet level, as shown in *Figure 7.8*.

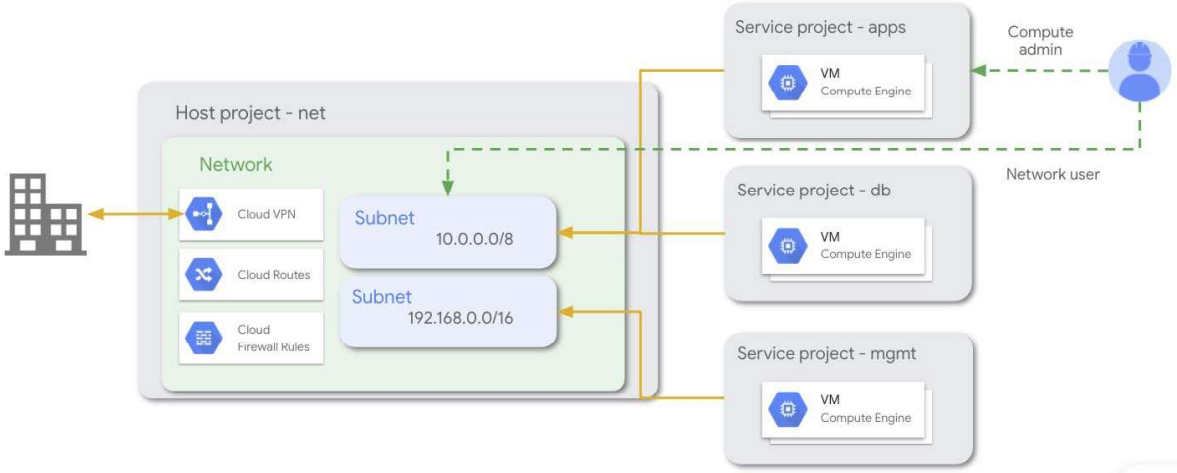

Figure 7.8 – Shared VPC network

Let us understand the IAM permissions for Shared VPC. You can use IAM roles for delegated administrator access. Organization admins can create Shared VPC Admin roles, and a Shared VPC Admin can further define and create Service Project VPC Admin roles. Shared VPC Admins have Compute Shared VPC Admin (`compute.xpnAdmin`) access and Project IAM Admin role (`resourcemanager.projectIAMAdmin`) permissions. A Service Project Admin can have service-project-wide scope and access to all host project subnets. The Shared VPC Admin can restrict access to particular subnets to make things more restrictive.

Shared VPC does have some limitations:

- You can have multiple Shared VPC networks in a host project, but you can only have one service project attached to a host project.
- You can have a maximum of 100 service projects attached to a host project.

Let us now look at how to configure IAM permissions before you create a Shared VPC network:

1. Before you can create the Shared VPC network, you need to make sure that the right IAM roles are assigned either at the organization level or at the folder level.

   We will now look at the steps to assign two roles; the first will be **Compute Shared VPC Admin** and the next is **Project IAM Admin**.

   Navigate to **IAM & admin** from the Google Cloud console navigation menu on the left and select **IAM**.

2. You must log in to the console as an organization administrator and then go to the **IAM** page to see the role.
3. Select your project, if not already selected. Refer to the following figure showing where to navigate to add the role for a new principal. In *Figure 7.9*, for the Shared VPC resource, **Project IAM Admin** and **Compute Network Viewer** roles have been assigned. Click on **ADD ANOTHER ROLE**, and you can then assign **Compute Shared VPC Admin**.

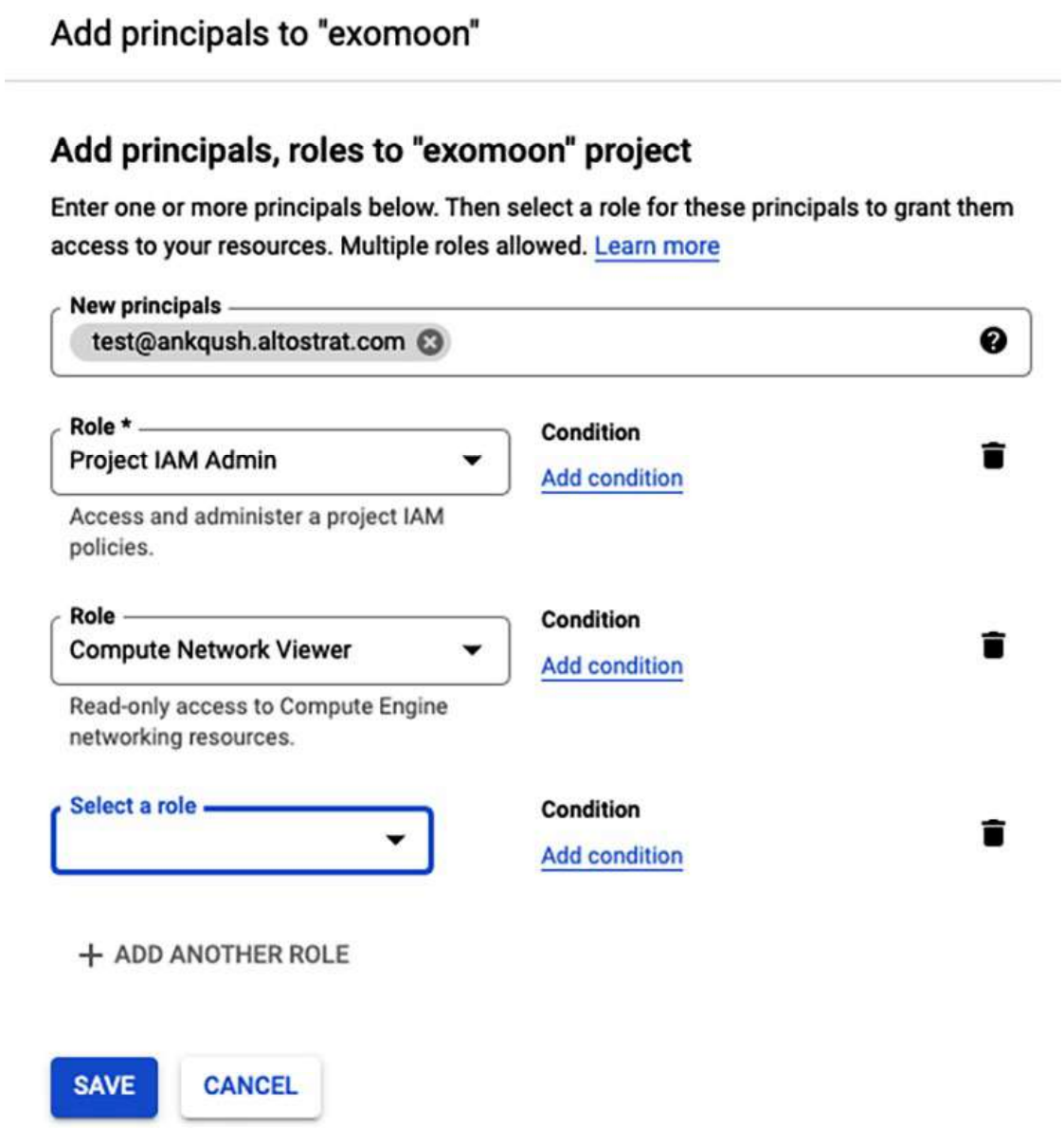

Figure 7.9 – Assign IAM roles for Shared VPC

4. Once you have completed the settings, click on **SAVE**.

Once the IAM roles have been assigned, we can now create the Shared VPC network in the host project:

1. Navigate to the **Shared VPC** page from the Google Cloud console.
2. Ensure that you are logged in as **Shared VPC Admin**.
3. From the project picker, select the Shared VPC host project.
4. Click **SET UP SHARED VPC**.

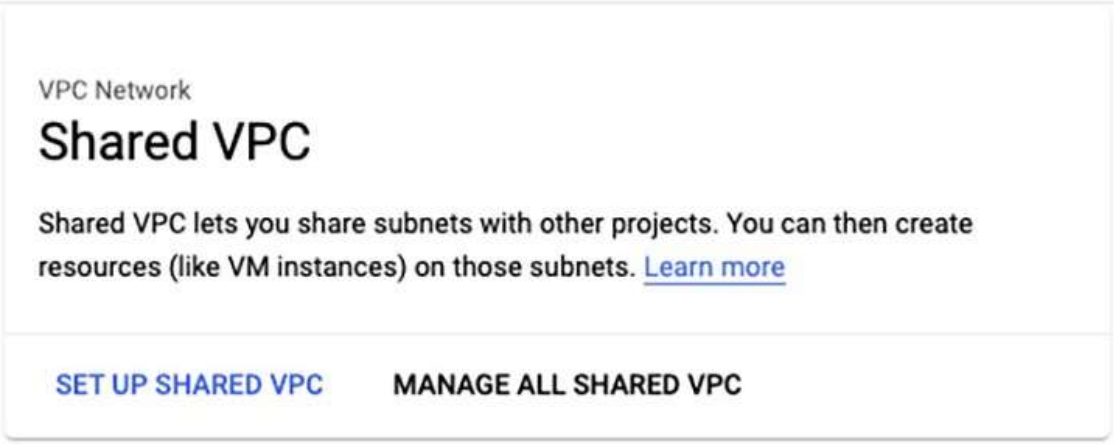

Figure 7.10 – SET UP SHARED VPC

5. We will now move on to the next page, where we will click **SAVE & CONTINUE** under **Enable host project**. Here you will select your existing project as your host project. Once done, we will then move on to configuring subnets in the next step.

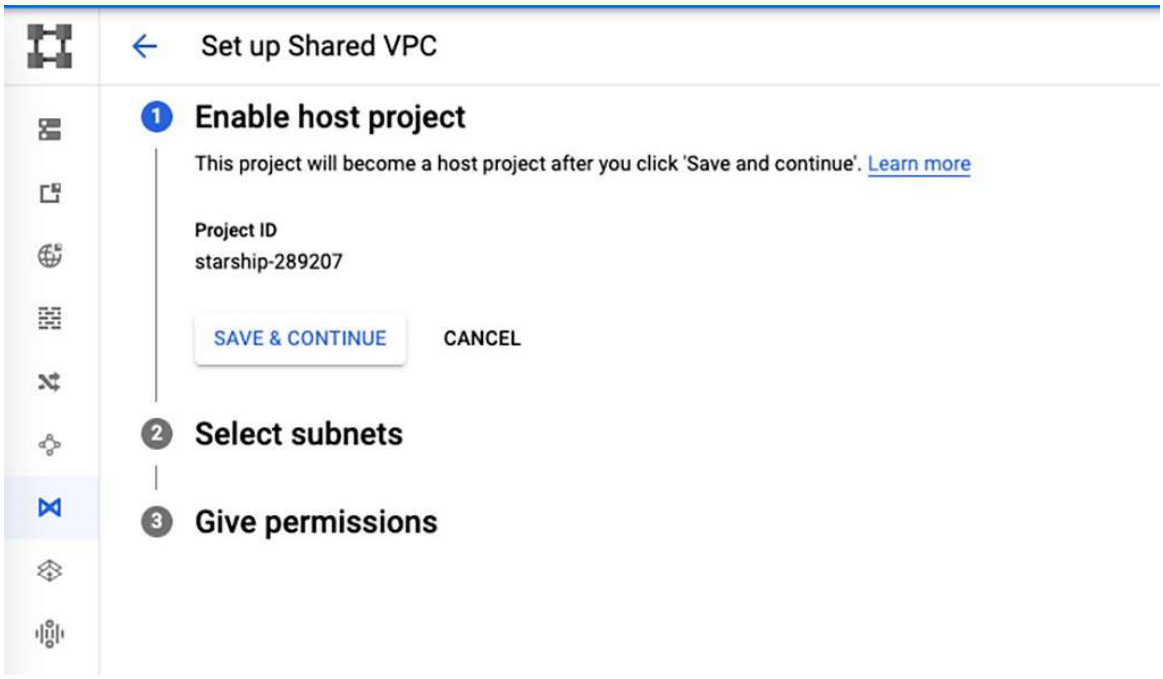

Figure 7.11 – Enable host project

6. There are two choices for selecting subnets in the **Select subnets** section. You can click **Share all subnets** if you want to share all the host project's VPC networks with the service projects and Service Project Admins listed below (project-level permissions). Alternatively, if you need to share some VPC network subnets with service projects and Service Project Admins, you can pick **Individual subnets** (subnet rights). Select the subnets you want to share first. Refer to *Figure 7.12* to see what options are available.