VM Manager also provides a nice graphical interface for visualization of the findings. Let us look at Rapid Vulnerability Detection.

Rapid Vulnerability Detection

Rapid Vulnerability Detection detects weak credentials, incomplete software installations, and other critical vulnerabilities that have a high likelihood of being exploited. The service automatically discovers network endpoints, protocols, open ports, network services, and installed software packages.

Rapid Vulnerability Detection findings are early warnings of vulnerabilities that Google recommends you fix immediately.

Here is a list of findings generated by Rapid Vulnerability Detection:

> **Note**
> Please refer to the Google Cloud documentation to find out the details of the findings and the remediation.

Weak credential findings:

- `WEAK_CREDENTIALS`

Exposed Interface findings:

- `ELASTICSEARCH_API_EXPOSED`
- `EXPOSED_GRAFANA_ENDPOINT`
- `EXPOSED_METABASE`
- `EXPOSED_SPRING_BOOT_ACTUATOR_ENDPOINT`
- `HADOOP_YARN_UNAUTHENTICATED_RESOURCE_MANAGER_API`
- `JAVA_JMX_RMI_EXPOSED`
- `JUPYTER_NOTEBOOK_EXPOSED_UI`
- `KUBERNETES_API_EXPOSED`
- `UNFINISHED_WORDPRESS_INSTALLATION`
- `UNAUTHENTICATED_JENKINS_NEW_ITEM_CONSOLE`

Software findings for remote code execution vulnerabilities:

- APACHE_HTTPD_RCE
- APACHE_HTTPD_SSRF
- CONSUL_RCE
- DRUID_RCE
- DRUPAL_RCE
- FLINK_FILE_DISCLOSURE
- GITLAB_RCE
- GoCD_RCE
- JENKINS_RCE
- JOOMLA_RCE
- LOG4J_RCE
- MANTISBT_PRIVILEGE_ESCALATION
- OGNL_RCE
- OPENAM_RCE
- ORACLE_WEBLOGIC_RCE
- PHPUNIT_RCE
- PHP_CGI_RCE
- PORTAL_RCE
- REDIS_RCE
- SOLR_FILE_EXPOSED
- SOLR_RCE
- STRUTS_RCE
- TOMCAT_FILE_DISCLOSURE
- VBULLETIN_RCE
- VCENTER_RCE
- WEBLOGIC_RCE

Now that we have looked at Rapid Vulnerability Detection, let us move on to web security vulnerabilities.

Web Security Scanner

WSS scans your App Engine, GKE, and Compute Engine web apps for security flaws. It crawls your apps, following all links that fall within the scope of your application's entry URLs, and tries to test as many user inputs and event handlers as possible. WSS only works with public URLs and IP addresses that are not behind a firewall.

App Engine standard and flexible environments, Compute Engine instances, and GKE resources are presently supported by WSS.

WSS works with your secure design and development procedures. It uses a combination of automated and manual techniques to scan web applications and services for common vulnerabilities such as SQL injection, cross-site scripting, and other security issues. It works by sending a series of requests to the target application or service and then analyzing the responses for possible vulnerabilities. WSS can also be used to conduct manual security reviews by providing detailed reports on potential issues. Additionally, WSS also provides remediation advice and steps to address any identified vulnerabilities.

The **OWASP Top Ten**, a publication that ranks and provides remedial assistance for the top 10 most critical online application security vulnerabilities, is supported by WSS.

There are two types of scans WSS supports:

- **Managed scans**: Managed scans detect and scan public web endpoints once a week automatically:
  - These scans do not employ authentication and perform GET-only queries, so no forms are submitted on live websites.
  - Managed scans are done independently of custom scans defined at the project level. Managed scan findings are automatically available on the SCC **Vulnerabilities** tab and related reports when you enable WSS as a service.

  A managed scan is especially useful without involving individual project teams, as you can utilize managed scans to centrally manage basic web application vulnerability detection for projects in your organization. You can work with those teams later to set up more comprehensive bespoke scans after the discoveries are made.
- **Custom scans**: WSS custom scans provide detailed information on application vulnerabilities such as obsolete libraries, cross-site scripting, and mixed content usage.

Here is a list of all the findings generated by WSS:

> **Note**
> For a description of these findings and remediation, look up the relevant Google Cloud documentation.

- `ACCESSIBLE_GIT_REPOSITORY`
- `ACCESSIBLE_SVN_REPOSITORY`
- `CACHEABLE_PASSWORD_INPUT`
- `CLEAR_TEXT_PASSWORD`
- `INSECURE_ALLOW_ORIGIN_ENDS_WITH_VALIDATION`
- `INSECURE_ALLOW_ORIGIN_STARTS_WITH_VALIDATION`
- `INVALID_CONTENT_TYPE`
- `INVALID_HEADER`
- `MISMATCHING_SECURITY_HEADER_VALUES`
- `MISSPELLED_SECURITY_HEADER_NAME`
- `MIXED_CONTENT`
- `OUTDATED_LIBRARY`
- `SERVER_SIDE_REQUEST_FORGERY`
- `SESSION_ID_LEAK`
- `SQL_INJECTION`
- `STRUTS_INSECURE_DESERIALIZATION`
- `XSS`
- `XSS_ANGULAR_CALLBACK`
- `XSS_ERROR`
- `XXE_REFLECTED_FILE_LEAKAGE`

While WSS is easy enough to use, there are some best practices that you should follow while using it. We will go over those now.

Best practices for WSS

You should use WSS with caution because it populates fields, pushes buttons, clicks links, and performs other user actions. This is especially true if you are scanning production resources. WSS may trigger features that alter the condition of your data or system, resulting in unfavorable outcomes. Here are some best practices:

- **Use a test environment for scanning**: Create a separate App Engine project and load your application and data there to create a test environment. When you upload your app using the Google Cloud CLI, you can provide the target project as a command-line argument.
- **Create a test account**: Create a user account with no access to sensitive data or potentially dangerous operations and use it to scan your app. When users log in for the first time, many applications need them to go through a unique process, such as accepting terms and creating a profile. A test account for an initial user may have different scan results than an established user account due to the different workflow. After the first-time flow is complete, scan using an account that is in the typical user state.
- **Deactivate some UI elements**: Apply the `no-click` CSS class to individual user interface elements that you do not wish to activate. Regardless of whether they are inline JavaScript, attached using `addEventListener`, or attached by specifying the appropriate event handler property, event handlers attached to this element are not enabled during crawling and testing.
- **Use backup data**: Before scanning, consider making a backup of your data.
- **Exclude URLs**: You can exclude URL patterns from being crawled or tested. See the following link for further details on the syntax: `https://packt.link/n1dwr`.
- **Scope of the scan**: Before you scan, check your app for any features that could affect data, users, or systems in ways that go beyond the scope of your scan.

So far, we have seen vulnerability detection findings when using SCC. Now we will look at the threat detection capabilities of SCC.

Threat detection

Google Cloud provides several types of threat detection via SCC Premium:

- Event Threat Detection
- Container Threat Detection
- VM Threat Detection
- Anomaly Detection

Let us start with ETD.

Event Threat Detection

Event Threat Detection (**ETD**) is a built-in feature of the SCC Premium tier that watches your Google Cloud environment in real time and detects threats within your systems. New detectors are added to ETD regularly to discover emerging threats at cloud scale.

ETD produces security findings by matching events in your Cloud Logging and Google Workspace log streams to known **indicators of compromise** (**IoCs**). IoCs, developed by internal Google security sources, identify potential vulnerabilities and attacks. ETD also detects threats by identifying known adversarial tactics, techniques, and procedures in your logging stream, and by detecting deviations from the historically observed behavior of your Google Cloud organization.

Here are some ETD features that you should be aware of:

- Project-specific logs are consumed when they become accessible by ETD. Cloud Logging records API calls and other operations that create, read, or modify resource configuration or metadata. Google Workspace logs keep track of user sign-ins to your domain and actions taken in the Google Workspace Admin console.
- ETD leverages log entries to swiftly detect threats. It uses Google's proprietary threat intelligence and detection technology to identify threats in near-real time.
- ETD also logs threats to SCC and Cloud Logging projects.
- You can use Cloud Functions to process data from Cloud Logging and Google Workspace Logging to automate responses to threats.
- You can also utilize Chronicle to explore findings. Chronicle is a Google Cloud service that lets you investigate threats and track them over time.

Now that you know what ETD is, let us see what it needs to detect threats.

Unsafe IAM changes

ETD finds external group members and examines each affected group's IAM roles using Cloud Audit Logs to see whether the groups have been granted sensitive responsibilities. This data is used to detect the following list of potentially dangerous modifications in privileged Google groups.

Unsafe changes in Google groups that involve high- or medium-sensitivity roles result in findings. The severity grade attributed to findings is influenced by the sensitivity of the roles:

- Billing, firewall settings, and logging are all controlled by high-sensitivity roles in businesses. Findings that correspond to these responsibilities are given a high severity rating.

- Editing permissions on Google Cloud resources, as well as viewing and executing permissions on data storage services that often hold sensitive data, are available to principals in medium-sensitivity positions. The severity of discoveries is determined by the resource:
  - Findings are categorized as *High* severity if medium-sensitivity roles are awarded at the organizational level
  - Findings are categorized as *Medium* severity if medium-sensitivity responsibilities are provided at lower levels in your resource hierarchy (folders, projects, and buckets, for example)

Look at Google Cloud's documentation for information on all highly privileged IAM role changes. The link is provided at the end of this chapter.

Let us look at the types of logs ETD needs to be able to detect threats.

Various log types for ETD

Admin Activity logs, which are part of Cloud Audit Logs, are automatically consumed by ETD (see the following list). Admin Activity logs are created automatically and do not require any configuration. Furthermore, turning on extra logs, which the service analyzes to detect specific threats, improves the performance of ETD. Here are various logs that you will need to enable specifically:

- SSH logs/syslog
- Data Access logs (part of Cloud Audit Logs)
- VPC flow logs
- Cloud DNS logs
- Firewall rules logs
- Cloud NAT logs
- **GKE** Data Access audit logs
- HTTP(S) load balancing backend service logs
- MySQL Data Access audit logs
- PostgreSQL Data Access audit logs
- Resource Manager Data Access audit logs
- SQL Server Data Access audit logs

> **Note**
> With frequent sampling and short aggregation periods, ETD works best. There may be a delay between the occurrence and detection of an event if you use lower sample rates or longer aggregation intervals. This lag can make evaluating any malware, cryptomining, or phishing traffic more difficult.

Before you turn on a log for ETD to scan, make sure you understand how Cloud Logging charges for the log data. After turning on a log, monitor the log for a couple of days to ensure that the log doesn't incur any unexpected Cloud Logging costs.

Although the scanning of logs by ETD does not incur any additional costs, depending on the volume of log data that your organizations and projects produce, Cloud Logging may charge you for the ingestion and storage of the log data.

Here are various Admin Activity logs that ETD scans automatically – you do not need to turn on or configure these:

- BigQueryAuditMetadata Data Access logs
- Cloud DNS Admin Activity audit logs
- GKE Admin Activity audit logs
- IAM Admin Activity audit logs
- MySQL Admin Activity logs
- PostgreSQL Admin Activity logs
- SQL Server Admin Activity logs
- VPC Service Controls Audit logs

Now that we have looked at Google Cloud logs, let us quickly understand the Workspace logs that ETD uses.

Google Workspace audit logs

These are enabled and maintained in your Google Workspace environment, although they must be shared with Google Cloud for ETD to access and detect Google Workspace threats:

- Login Audit logs (for Data Access audit logs)
- Admin Audit logs (for Admin Activity audit logs)

Now you know which logs ETD uses to detect threats, let us look at the findings ETD generates.

ETD findings

Here are some examples of findings generated by ETD:

- Evasion: Access from Anonymizing Proxy
- Exfiltration: BigQuery Data Exfiltration
- Exfiltration: Cloud SQL Data Exfiltration
- Brute Force: SSH
- Credential Access: External Member Added To Privileged Group
- Credential Access: Privileged Group Opened To Public
- Credential Access: Sensitive Role Granted To Hybrid Group
- Crypto Mining
- Initial Access: Log4j Compromise Attempt
- Active Scan: Log4j Vulnerabilities To RCE
- Leaked Credentials
- Malware
- Outgoing DOS
- Persistence: IAM Anomalous Grant
- Persistence: New Geography
- Persistence: New User Agent
- Phishing
- Service Account Self-Investigation
- Compute Engine Admin Metadata detections
- Persistence: Compute Engine Admin Added Startup Script

Refer to the Google Cloud documentation for a list of all findings and how to build a response.

Google Workspace threat detection

Let us look at various detectors available in Google Workspace now:

- Initial Access: Disabled Password Leak
- Initial Access: Suspicious Login Blocked
- Initial Access: Account Disabled Hijacked

- Impair Defenses: Two-Step Verification Disabled
- Initial Access: Government-Based Attack
- Persistence: SSO Enablement Toggle
- Persistence: SSO Settings Changed
- Impair Defenses: Strong Authentication Disabled

This concludes the section on ETD. Let us look at KTD now.

Container Threat Detection

KTD is another built-in service within SCC Premium that supports threat detection on the GKE platform. Please verify the version of the GKE cluster you are running to make sure it is supported. Refer to the Google Cloud documentation for supported versions. KTD gathers and analyzes low-level observable behavior in the guest kernel of your containers to generate insights.

Here are the findings generated by KTD:

- **Added Binary Executed**: A binary that was not part of the original container image was executed. Attackers commonly install exploitation tooling and malware after the initial compromise.
- **Added Library Loaded**: A library that was not part of the original container image was loaded. Attackers might load malicious libraries into existing programs to bypass code execution protections and hide malicious code.
- **Malicious Script Executed**: A machine learning model identified an executed Bash script as malicious. Attackers can use Bash to transfer tools and execute commands without binaries.
- **Reverse Shell Attack**: A process started with stream redirection to a remote-connected socket. With a reverse shell, an attacker can communicate from a compromised workload to an attacker-controlled machine. The attacker can then command and control the workload to perform desired actions, for example, as part of a botnet.

Now that we understand how KTD works, let us move on to learning how VM Threat Detection works.

VM Threat Detection

VM Threat Detection (**VMTD**) is a built-in feature of SCC Premium that detects threats at the hypervisor level. VMTD detects coin mining software, which is one of the most typical forms of software found in exploited cloud environments.

VMTD is a part of the SCC Premium threat detection suite, and it is meant to work alongside ETD and KTD. This feature is now in **general availability** (**GA**) as of July 2022. The certification exam will not have any questions on it at the time of publication of this book; it may be added in the future.

The service executes scans from the hypervisor into the guest VM's live memory regularly without halting the guest's activity. VMTD analyzes information about software running on VMs, such as a list of application names, per-process CPU usage, hashes of memory pages, CPU hardware performance counters, and information about executed machine code, using Google Cloud's threat detection rules to see whether it matches known crypto mining signatures.

Because VMTD operates from outside the guest VM instance, it does not require guest agents or specific guest OS configuration, and it is immune to complex malware countermeasures. Inside the guest VM, no CPU cycles are used, and network access is not required. Signatures do not need to be updated, and the service does not need to be managed. VMTD is not supported on confidential computing since it operates with hardware encryption of memory.

The capabilities of Google Cloud's hypervisor are required for VMTD; it cannot be used in on-premises or other public cloud settings.

Here are the threats it will alert you to:

- **Execution: Cryptocurrency Mining Hash Match**: Matches memory hashes of running programs against known memory hashes of cryptocurrency mining software.
- **Execution: Cryptocurrency Mining YARA Rule**: Matches memory patterns, such as proof-of-work constants, known to be used by cryptocurrency mining software.
- **Execution: Cryptocurrency Mining Combined Detection**: Combines multiple categories of findings detected within a one-hour period. The threats are rolled into a single finding.

You have seen how VMTD can be used; now let us look at the last topic of threat detection, anomaly detection.

Anomaly detection

Anomaly detection is a built-in feature that uses external behavior signals to detect anomalies. It shows granular details about security abnormalities found in your projects and VM instances, such as potential credential leaks and currency mining. When you subscribe to the SCC Standard or Premium tier, anomaly detection is turned on immediately, and the results are displayed on the SCC dashboard.

This feature is available for the Standard and Premium versions of SCC. Here are the findings detected by anomaly detection. They are generated in the following two categories:

- **Potential for compromise**:
  - `account_has_leaked_credentials`: Credentials for a Google Cloud service account are accidentally leaked online or are compromised.
  - `resource_compromised_alert`: Potential compromise of a resource in your organization.

- **Abuse scenarios**:
  - `resource_involved_in_coin_mining`: Behavioral signals around a VM in your organization indicate that a resource might have been compromised and could be getting used for crypto mining.
  - `outgoing_intrusion_attempt`: One of the resources or Google Cloud services in your organization is being used for intrusion activities, such as an attempt to break into or compromise a target system. These include SSH brute force attacks, port scans, and FTP brute force attacks.
  - `resource_used_for_phishing`: One of the resources or Google Cloud services in your organization is being used for phishing.

Now that we understand various abilities of threat detection, let us move on to look at the SCC feature for compliance monitoring.

Continuous compliance monitoring

In a regulated organization, compliance takes precedence as it is mandated by regulations; however, compliance is now generally enforced by all security-conscious organizations, regardless of regulations. SCC provides the ability to continuously monitor your Google Cloud compliance posture by doing the following:

- Identifying compliance violations in your Google Cloud assets helps you resolve them by following actionable suggestions
- Reviewing and exporting compliance reports to ensure all your resources are meeting their compliance requirements
- Supporting compliance standards, such as these:
  - **Payment Card Industry Data Security Standard** (**PCI DSS v3.2.1**)
  - **International Organization for Standardization** (**ISO 27001**)
  - **National Institute of Standards and Technology** (**NIST 800-53**)
  - **Center for Internet Security** (**CIS**) 1.0 and 1.1 benchmarks

Now let us look at how SCC supports these standards.

CIS benchmarks

SCC supports the following CIS benchmarks for Google Cloud. CIS 1.0 and CIS 1.1 are still supported, but eventually, they will be deprecated. It is recommended to use or transition to using the latest benchmark, CIS 1.2:

- CIS Google Cloud Computing Foundations Benchmark v1.2.0 (CIS Google Cloud Foundation 1.2)
- CIS Google Cloud Computing Foundations Benchmark v1.1.0 (CIS Google Cloud Foundation 1.1)
- CIS Google Cloud Computing Foundations Benchmark v1.0.0 (CIS Google Cloud Foundation 1.0)

Additional standards

The PCI DSS and the OWASP Foundation do not supply or approve additional compliance mappings, which are included for reference only. To manually check for these violations, consult the PCI Standard 3.2.1 (PCI-DSS v3.2.1), OWASP Top Ten, NIST 800-53, and ISO 27001.

Note that this feature is exclusively for customers to use to check for violations of compliance regulations. The mappings are not intended to be used as the foundation for, or as a substitute for, an audit, certification, or report of compliance with any regulatory or industry benchmarks or standards for your products or services.

So far, we have seen how SCC generates the finding by using various modules. Let us now move on to see how we can export the findings for analysis. The findings can also be sent for remediation in SOAR tools such as Google Chronicle SOAR.

Exporting SCC findings

SCC allows you to export SCC data, including assets, findings, and security marks to another system of your choice:

- Exports of current discoveries, assets, and security markings on a one-time basis
- Continuous exports, which automatically export new discoveries to Pub/Sub SCC for Premium users, and allow you to export data using the SCC API or the Google Cloud console
- You may also export your results to BigQuery

Let us look at these now.

One-time exports

You can manually transmit and download current and historical findings and assets in JSON or JSONL format via one-time exports using the Google Cloud console. You can upload data to a Cloud Storage bucket and then download it to your local machine. You need to take security measures to ensure all locally stored findings are stored safely.

Exporting data using the SCC API

Using the SCC API, you may export assets, results, and security marks to a Cloud Storage bucket. The API results for discoveries or assets can be retrieved or exported once they have been listed:

- The `ListFindings` and `ListAssets` API methods are used to list findings or assets with any connected security markings. The methods return assets or findings in JSON format, complete with all their characteristics, attributes, and related marks. If you need data in a different format for your application, you will need to develop additional code to convert the JSON output.
- The `GroupAssets` or `GroupFindings` methods are utilized if the `groupBy` field is set to a value. The `GroupAssets` and `GroupFindings` methods deliver a list of assets or findings for an organization, grouped by properties.

Continuous exports

SCC Premium clients can use continuous exports to automate the process of exporting SCC findings to Pub/Sub:

- New discoveries are immediately exported to selected Pub/Sub topics in near-real time when they are written, allowing you to integrate them into your existing process
- Using the SCC API, you may set up Pub/Sub finding notifications in SCC
- To set up Pub/Sub topics, construct finding filters, and create `NotificationConfig` files (files that contain notification configuration options), you must utilize the Google Cloud CLI

Now let us go over how to use Pub/Sub for continuous exports.

Configuring Pub/Sub exports

Continuous exports let you automate the export of all future findings to Pub/Sub or create filters to export future findings that meet specific criteria. You can filter findings by category, source, asset type, security marks, severity, state, and other variables. Let us see how to set up an export now.

Creating continuous exports

Your organization can create a maximum of 500 continuous exports. To create an export for Pub/Sub, do the following:

1. Go to the SCC **Findings** page in the Cloud console. In the **Filter** field, select the attributes, properties, or security marks you want to use to filter findings and enter the desired variables. A blank filter is evaluated as a wildcard and all findings are exported.
2. Click **Export**, and then, under **Continuous**, click **Pub/Sub**.
3. Review your filter to ensure it is correct and, if necessary, return to the **Findings** page to modify it.
4. Under **Continuous export name**, enter a name for the export.
5. Under **Continuous export description**, enter a description for the export.
6. Under **Export to**, select a project for your export.
7. Under **Pub/Subtopic**, select the topic where you want to export findings. To create a topic, do the following:

    I. Select **Create a topic**.

    II. Fill out **Topic ID**, and then select other options as needed.

    III. Click **Create Topic**.

8. Click **Save**. You will see a confirmation and are returned to the **Findings** page.
9. Follow the guide to create a subscription for your Pub/Sub topic.

The Pub/Sub export configuration is complete. To publish notifications, a service account is created for you in the form of `service-org-ORGANIZATION_ID@gcp-sa-scc-notification.iam.gserviceaccount.com`. This service account is automatically granted the `securitycenter.notificationServiceAgent` role at the organization level. This service account role is required for notifications to function.

BigQuery is a popular tool for the analysis of vast amounts of data. Now let us look at how to export findings to BigQuery for analysis.

Exporting findings to BigQuery

Any new discoveries written to SCC can be immediately exported to a BigQuery table. The data can then be integrated into current workflows and a unique analysis can be created. You can use this functionality to export findings based on your needs at the organization, folder, and project levels.

Because it is fully managed and does not need manual procedures or custom code, this feature is the ideal solution for exporting SCC findings to BigQuery.

Here are a few points that you need to know before you can export your findings to BigQuery:

- You create an export configuration to export findings to BigQuery. You can create export configurations at the project, folder, or organization level. For example, if you want to export findings from a project to a BigQuery dataset, you create an export configuration at the project level to export only the findings related to that project. Optionally, you can specify filters to export certain findings only.
- You can create a maximum of 500 export configurations to BigQuery for your organization. You can use the same dataset for multiple export configurations. If you use the same dataset, all updates will be made to the same findings table.
- When you create your first export configuration, a service account is automatically created for you. This service account is required to create or update the findings table within a dataset and to export findings to the table. It has the form `service-org-ORGANIZATION_ID@gcp-sa-scc-notification.iam.gservicaccount.com` and is granted the `BigQuery Data Editor (roles/bigquery.dataEditor)` role at the BigQuery dataset level.

Now let us go over the steps for exporting the SCC findings to BigQuery:

1. Go to the Google Cloud console.
2. Select the project for which you enabled the SCC API.
3. Click **Activate Cloud Shell**.
4. To create a new export configuration, run the following command:

```
gcloud scc bqexports create BIG_QUERY_EXPORT \
  --dataset=DATASET_NAME \
  --folder=FOLDER_ID | --organization=ORGANIZATION_ID |
--project=PROJECT_ID \
  [--description=DESCRIPTION] \
  [--filter=FILTER]
```

   Replace the following in the preceding command:

   - Replace `BIG_QUERY_EXPORT` with a name for this export configuration.
   - Replace `DATASET_NAME` with the name of the BigQuery dataset; for example, `projects/<PROJECT_ID>/datasets/<DATASET_ID>`.
   - Replace `FOLDER_ID`, `ORGANIZATION_ID`, and `PROJECT_ID` with the name of your folder, organization, or project. You must set one of these options. For folders and organizations, the name is the folder ID or the organization ID. For projects, the name is the project number or the project ID.

- Replace `DESCRIPTION` with a human-readable description of the export configuration. This variable is optional.
- Replace `FILTER` with an expression that defines what findings to include in the export. For example, if you want to filter on the `XSS_SCRIPTING` category, type `"category=\"XSS_SCRIPTING\"`. This variable is optional.

5. To verify the details of the export configuration, run the following command:

```
gcloud scc bqexports get BIG_QUERY_EXPORT \  --folder=FOLDER_ID
| --organization=ORGANIZATION_ID | --project=PROJECT_ID
```

Replace the following:

- Replace `BIG_QUERY_EXPORT` with the name for this export configuration.
- Replace `FOLDER_ID`, `ORGANIZATION_ID`, and `PROJECT_ID` with the name of your folder, organization, or project. You must set one of these options. For folders and organizations, the name is the folder ID or the organization ID. For projects, the name is the project number or the project ID.

You should see the findings in your BigQuery dataset about 15 minutes after you create the export configuration. After the BigQuery table is created, any new and updated findings that match your filter and scope will appear in the table in near-real time. Check out the Google Cloud documentation for useful queries to view the findings.

We have seen how to export findings via Pub/Sub. These findings can be sent to SOAR tools to build an automated response. We will quickly go over that workflow now.

Automating a findings response

Google provides procedures for the following four types of SOAR products for exporting SCC alerts and findings:

- Palo Alto Cortex XSOAR
- Elastic Stack
- Splunk
- IBM QRadar

However, you can set up an integration using Pub/Sub to any other product if it can ingest and parse the Pub/Sub message.

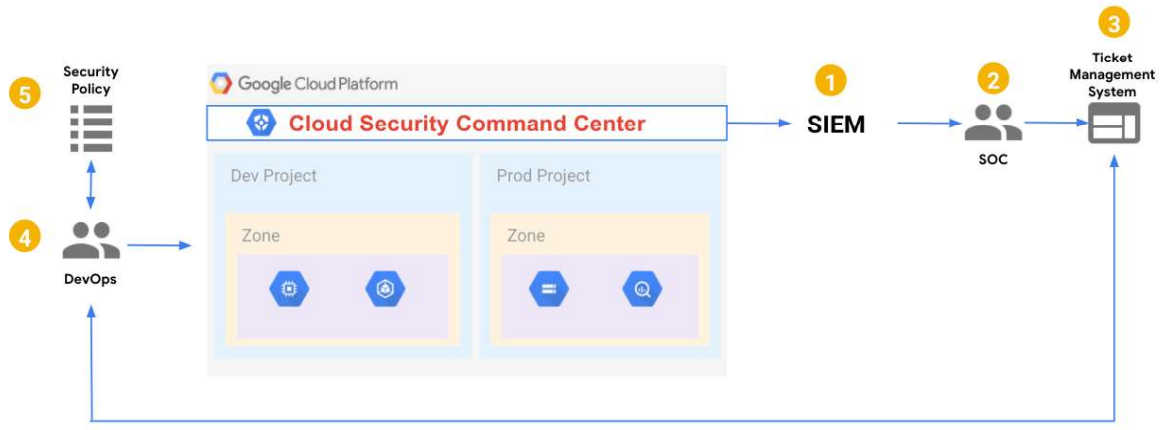

Figure 14.2 – Automating SCC response

Figure 14.2 shows an architecture of a simple workflow to automate a response based on SCC alerts and findings. The steps in the workflow are listed as follows. Each of these steps represents an action that can be taken either manually or automated. Understanding the various categories of threats and vulnerability findings is critical before building such a workflow.

Let us quickly run through these steps:

1. SCC alerts and findings are exported to the SIEM tool of your choice. As you saw in the previous section, you can export alerts and findings using continuous exports.
2. An SIEM alert is generated based on a pre-defined criterion. Your SIEM tool will have some logic for how to interpret the alert categories and what severity they are.
3. The SOC analyst collects information, understands the context, and classifies the incident.
4. The SOC analyst opens an IT service management ticket and assigns it to the appropriate operations team.
5. The operations team remediates the issue or contacts the project team that owns the asset that generated the alert for remediation.
6. If the alert is new, the security team creates preventative controls and updates the security policy.

Summary

In this chapter, we went over several capabilities of SCC. We learned how to use CAI, set up an export to BigQuery, and run SQL queries to understand your environment. We also went over how to detect security misconfigurations using SHA, VM Manager, WSS, and Rapid Vulnerability Detection. These are all critical capabilities before a security misconfiguration vulnerability becomes a threat. We learned about the threat detection capabilities of SCC in the form of ETD, CTD, VMTD, and anomaly detection. We also covered continuous compliance monitoring to understand how to apply industry standards to your cloud environment. Finally, we explored a simple architecture pattern for alerting.

In the next chapter, we will cover container security and look at how security measures are taken to protect containers and the applications and data that reside in them. This includes preventing unauthorized access and mitigating the risk of malicious code or attacks, as well as ensuring that your containers are configured in accordance with industry best practices.

Further reading

For more information on Google Cloud Security Command Center, refer to the following links:

- Event Threat Detection rules: `https://packt.link/Oeyqd`
- Sensitive IAM roles and permissions: `https://packt.link/5V1TK`
- Useful BigQuery queries for SCC findings: `https://packt.link/n8rwU`
- Creating charts in Google Looker Studio for findings: `https://packt.link/esxM2`

15
Container Security

In this chapter, we will look at container security. Container security is critical for today's enterprises because containers have become the go-to technology for deploying applications in modern IT environments. As a result, they have become a target for attackers who seek to exploit vulnerabilities in container infrastructure and applications to gain access to sensitive data or cause harm to the organization. Without proper security measures in place, containers can create significant risks for enterprises, including data breaches, system downtime, and compliance violations. By prioritizing container security and implementing best practices, enterprises can protect their applications, data, and infrastructure from cyber threats and ensure the safe and secure deployment of their workloads. Container security is considered a critical aspect of cloud security. It is a broad topic, so we will try to cover it from the exam point of view.

In this chapter, we will cover the following topics:

- Overview of containers
- Container basics
- Kubernetes and Google's GKE platform
- Threats and risks in containers
- GKE security features such as namespaces, RBAC, and service meshes
- Container image security
- Container vulnerability scanning
- Binary authorization
- Cluster certification authority
- Container security best practices

Overview of containers

A container is a lightweight, standalone executable package that contains everything needed to run an application, including the code, runtime, libraries, and dependencies. Containers are designed to be easily portable between different computing environments, making them an ideal solution for modern application deployment. Everything at Google runs in containers, from Gmail to YouTube to Search. Development teams can now move quickly, distribute software efficiently, and operate at unprecedented scale thanks to containerization.

Containers come with security advantages inherent to their architecture:

- Containers are short-lived and frequently re-deployed
- Containers are intentionally immutable; a modified container is a default security alert
- Good security defaults are one-line changes; setting secure configurations is easy
- With isolation technologies, you can increase security without adding resources

Google invests massively in container security. Here is Google's container security philosophy:

- **Prioritize security by default**: Google believes in giving users the power of Kubernetes without making them security experts.
- **Integrate the best of Google Cloud Platform security**: Integrations with IAM, audit logging, VPCs, encryption by default, and Security Command Center.
- **Strong security culture**: Google dedicates a large amount of resources to securely designing and running systems. Security is an integral part of Google's culture, products, and operations.

But before we dive deeper into container security, let us first understand what containers are.

Container basics

Traditionally, applications used to be deployed on dedicated servers. To run an application, you would do the following:

1. Purchase hardware.
2. Install the OS.
3. Install dependencies.
4. Deploy application code.
5. Make sure the application is the same across all environments.

This took a lot of time and resources to deploy and maintain. It was not portable and was difficult to scale. VMware popularized running multiple servers and operating systems (OSs) on the same hardware using a hypervisor. Each virtual machine (VM) has its own dedicated resources, including memory, CPU, and storage, which are allocated by the hypervisor that manages them. VMs are isolated from each other and from the host machine, providing greater security and flexibility but also requiring more resources and longer startup times. Containers share the host machine's OS kernel and use containerization technology to isolate the application from other processes running on the host. Because they don't need to emulate an entire OS, containers are much smaller and faster to start up than VMs.

Now let us see how containers are structured.

What are containers?

Containers are a method of *packaging an application executable and its dependencies* (runtime, system tools, system libraries, configuration, and so on) and running the package as a set of resource-isolated processes.

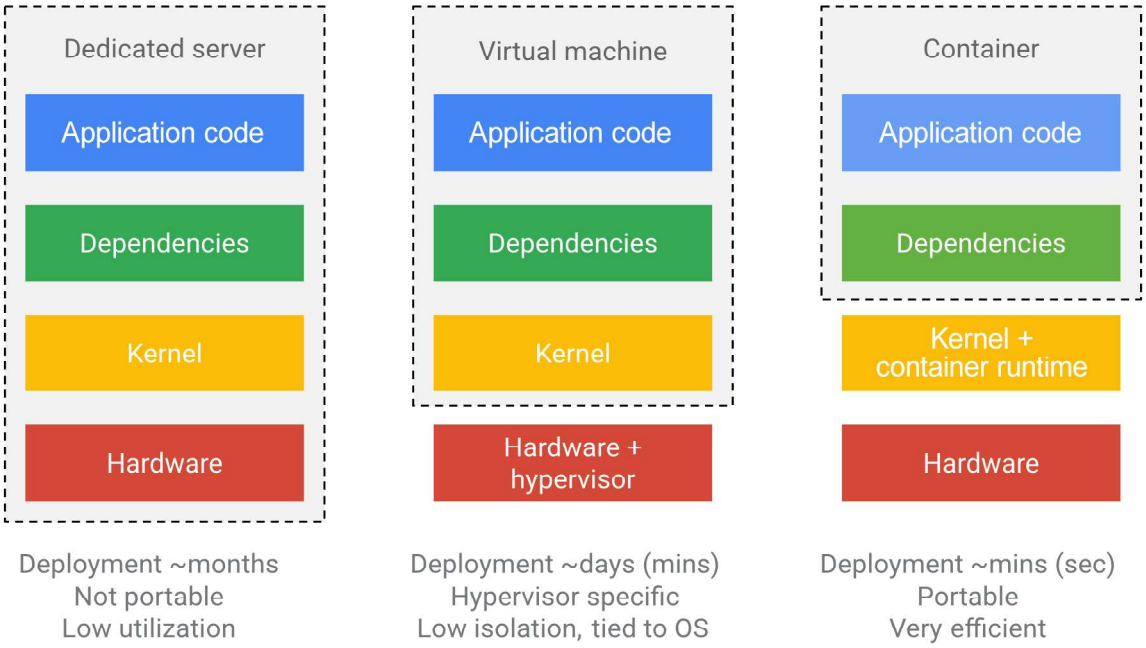

Figure 15.1 – Container structure

Containers raise the abstraction one more level and virtualize the OS. They are extremely portable and can be run locally or in the cloud without any changes. Lightweight containers do not carry a full OS and can be packed tightly onto available resources. Fast startup is no more than starting a process on the OS.

Containers use the concept of a **layered filesystem**. The base image layers are all read-only. Updates are made to a small read-write layer built on top of a base image. Each container has its own writable container layer. All changes are stored in this layer.

Multiple containers share the same underlying read-only image. Any changes to lower levels are first copied to the container layer where changes are then overlaid only for that container. The base image is shared across containers, taking less disk space and enabling fast container start times.

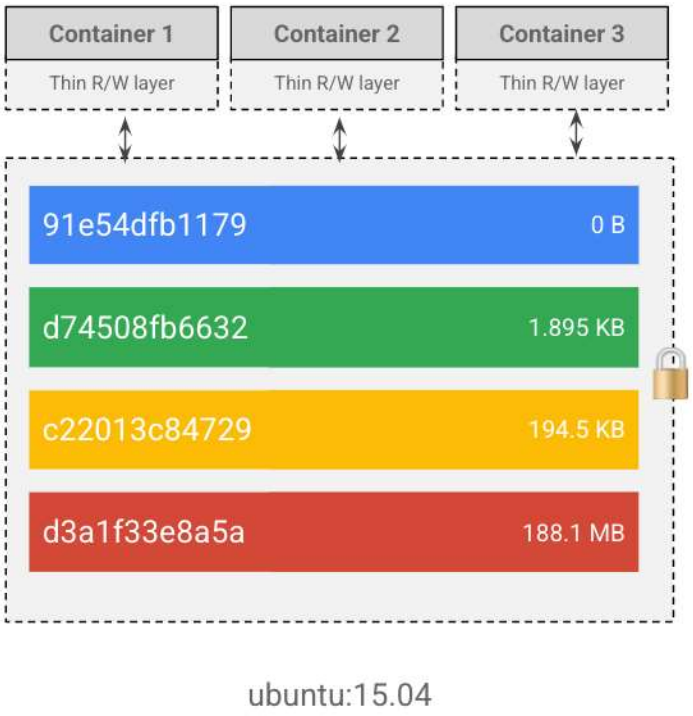

Figure 15.2 – Container layers

Let us go over some terminology in the container space:

- **Container image**: A container image is created by assembling the application code and dependencies into a single package that can be easily distributed and deployed. Once an image is created, it can be stored in a container registry, such as Docker Hub or Google Container Registry, and then used to spin up one or more containers on a Kubernetes cluster.
- **Container**: A container running in a pod is an instantiated container image.
- **Container host**: The container host is the computer that runs the containerized processes, commonly called containers.
- **Registry server**: A registry server is just a fancy file server where container images are stored.

- **Container orchestrator**: This is software that manages containers. Google GKE, AWS EKS, Azure AKS, and Red Hat OpenShift are examples of container orchestrators. A container orchestrator serves two purposes:
  - Container workloads are dynamically scheduled in a cluster of machines.
  - It uses a uniform application definition file to set up the cluster.
- **Kubernetes**: Kubernetes is an open source container orchestrator, developed by Google.
- **Namespaces**: Think of a namespace as a virtual mechanism to group containers within your Kubernetes cluster. You can have multiple namespaces inside a single cluster.
- **Docker**: Docker is the container runtime that builds, runs, and monitors containers. Other runtimes include LXC, systemd-nspawn, CoreOS rkt, and others.
- **Cgroups**: Control groups, or cgroups, is a feature of Linux that constrains the resources (CPU, memory, and so on) allocated to processes. The kubelet and the underlying runtime, such as Docker, use cgroups to enforce resource management for pods and containers.
- **Container registry**: A container registry is a repository that holds container images not yet deployed. Cloud providers including Google will provide a registry service in the cloud. For example, Google has Artifact Registry and Container Registry (deprecated) for container images.

Now that we understand the basic terminology of containers, let us understand the advantages of using containers.

Advantages of containers

There are many advantages of running containerized applications. Here are some:

- **Separation of code and computing**:
  - Consistency across dev, test, and production
  - Consistency across bare-metal, VMs, and the cloud
  - No more *it worked on my computer*
- **Packaged applications**:
  - Agile application creation and deployment
  - Containers aid massively in a Continuous Integration/Delivery/DevOps process
- **Conducive for microservices**:
  - Container deployments are introspectable
  - Isolated/loosely coupled, conducive to distributed systems, and elastic

- **Faster time to market**:
  - Small features can be rolled out multiple times a day rather than waiting weeks to deploy
  - Containers are easy to roll back, so if a feature fails in canary deployment, it can be quickly rolled back without impacting customers

Now that you understand the basics of containers and why to use containers, we will look at how to run containers in a scalable fashion using a container orchestrator such as Kubernetes.

What is Kubernetes?

Kubernetes, also known as **K8s**, is an open source system for automating the deployment, scaling, and management of containerized applications. The name Kubernetes originates from Greek, meaning helmsman or pilot. In simple terms, think of K8s as the orchestrator for your container fleet. It tracks how many containers are needed, which one is performing well, and how to direct your traffic, among other things.

Here are some features provided by K8s:

- **Load balancing and service discovery**: Kubernetes exposes a container using an independent IP address or a DNS name. Kubernetes may load balance and spread the traffic to keep the deployment stable.
- **Storage management**: Kubernetes can allow you to mount storage, also called volume, that containers in the pods can read and write to; for example, on GKE you can mount volumes such as **emptyDir**, **ConfigMap**, **Secret**, and so on.
- **Rollouts and rollbacks**: Kubernetes does an automated rollout and rollback for you. All you need to do is define the desired state of your application.
- **Packing**: Bin packing is done automatically for you. Bin packing is a scheduling strategy used by the Kubernetes scheduler to optimize the allocation of resources across nodes in a cluster. The goal of bin packing is to maximize the utilization of available resources while minimizing waste.
- **Self-healing**: This is a feature of Kubernetes implemented as auto-repair in GKE. Auto-repair makes periodic checks on the health of cluster nodes. If a node fails a health check, Kubernetes will try to repair that node.
- **Management of secrets and configurations**: Kubernetes will let you deploy secrets, such as OAuth tokens, passwords, and so on, securely. These are stored unencrypted in the K8S database. Secrets are kept separately from the pods that use them. That way, you do not need to embed secrets within your application code. GKE provides many advanced features for secret management.

Now that you understand what Kubernetes is, we should also understand what Kubernetes is not:

- Kubernetes doesn't impose restrictions on supported applications. Kubernetes promises to serve stateless, stateful, and data-processing workloads. Anything that runs in a container should run well on Kubernetes.
- Kubernetes does not build or deploy source code. CI/CD procedures are driven by organizational cultures and preferences and technical constraints.
- Kubernetes does not include application-level services such as middleware (message buses), data processing frameworks (Spark), databases (MySQL), caches, or cluster storage systems (Ceph). Applications running on Kubernetes can access such components via portable interfaces such as the Open Service Broker API. The Kubernetes **Open Service Broker** (**OSB**) is a component that allows Kubernetes users to easily provision and manage services from a variety of providers.
- Kubernetes has no configuration language/system (for example, Jsonnet). It exposes a declarative API that can target any declarative specification.
- Kubernetes has no extensive machine configuration, maintenance, management, or self-healing systems.
- Kubernetes isn't just an orchestration platform. Orchestration is the execution of a defined workflow: first, do A, then B, then C. Kubernetes, on the other hand, is a collection of *independent, composable control processes that continuously strive to achieve the desired state*. So, it does not matter how you go from state A to C. It also removes the need for centralized control. This makes the system more powerful, robust, resilient, and extensible.

Everything at Google runs on containers, such as Gmail, Search, Maps, and even **Google Cloud Platform** (**GCP**). Google is said to deploy over 4 billion containers every week.

Now let us look at Google Cloud's managed Kubernetes platform, called GKE.

GKE

Google Kubernetes Engine (**GKE**) is a managed platform for Kubernetes on Google Cloud. It provides the following additional features on top of K8s:

- Kubernetes clusters as a service
- It runs Kubernetes on GCE
- It is integrated with GCP
- It supports heterogeneous and multi-zone clusters
- It manages Kubernetes (auto-upgrades, scaling, healing, monitoring, backup, and so on)

Let us now look at the architecture of GKE.

GKE architecture

In GKE, the master is the control plane that manages the entire lifecycle of a cluster. The cluster master is responsible for deciding what runs on all the cluster's nodes and acts as a unified endpoint that provides the Kubernetes API Server for the cluster. The project that holds the control plane is managed by Google for you. You do not get access to this project.

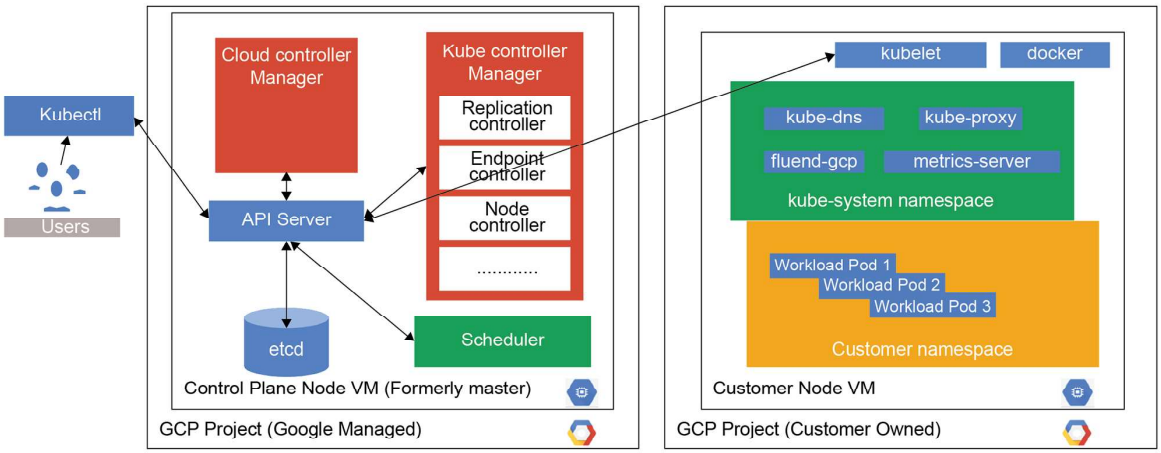

Figure 15.3 – GKE architecture

Here are the key components of GKE:

- **Control plane**: The control plane is the central management component of GKE. It includes various services that handle cluster management, scheduling, and communication between different components. The control plane manages the state of the cluster and ensures that the desired configuration is maintained. The control plane is managed by Google in a Google managed project. Customers do not have access to this project. The control plane in GKE is designed for high availability and fault tolerance. It runs across multiple compute instances (VMs) distributed across different availability zones within a region. The control plane consists of several master components, including the following:
  - **kube-apiserver**: This provides the API endpoint for interacting with the cluster. It handles authentication, authorization, and validation of API requests.
  - **etcd**: This is a distributed key-value store used to store the cluster's configuration data and state.
  - **kube-scheduler**: This assigns pods to nodes based on resource requirements, constraints, and availability.
  - **kube-controller-manager**: This runs various controllers responsible for maintaining the desired state of the cluster, handling replication, scaling, and other cluster-level operations.

    - **cloud-controller-manager**: This interacts with GCP APIs to manage resources such as load balancers, disks, and network routes.

- **Customer node VM**: Nodes are the worker machines in a GKE cluster in a customer project. Each node runs the Kubernetes components necessary for managing containers, such as the kubelet, kube-proxy, and container runtime (for example, Docker). Nodes are responsible for running the containerized applications and scaling resources based on demand.
- **Cluster**: A GKE cluster is a set of compute instances (nodes) that run containerized applications. The cluster is managed by the control plane and provides the underlying infrastructure to deploy and manage containers. It can span multiple availability zones for high availability and resilience.
- **Pods**: Pods are the smallest deployable units in Kubernetes. They encapsulate one or more containers and share the same network namespace and storage volumes. Pods are scheduled and managed by Kubernetes, and they provide an isolated environment for running applications.
- **Services**: Services provide network connectivity and load balancing for the pods within a GKE cluster. They define a stable endpoint that can be used to access the application running inside the pods. Services can be exposed internally within the cluster or externally to the internet.
- **Load balancers**: GKE integrates with Google Cloud Load Balancing to distribute traffic across multiple nodes and services. Load balancers help distribute incoming requests to pods within the cluster and provide scalability and high availability for applications.
- **Persistent storage**: GKE supports various options for persistent storage. It integrates with Google Cloud Storage, Persistent Disk, and other storage solutions to provide durable and scalable storage for applications running in the cluster. Persistent volumes can be dynamically provisioned and attached to pods as needed.
- **Container Registry**: GKE integrates with Google Container Registry, which is a private container image registry. It allows you to store and manage your container images securely. Container Registry provides an easy way to store, share, and deploy container images for your applications.
- **kubectl**: kubectl is a command-line tool used to interact with Kubernetes clusters. It is part of the official Kubernetes distribution and provides a convenient interface for managing and controlling Kubernetes resources. With kubectl, you can deploy and manage applications, inspect and modify cluster resources, view logs, execute commands within containers, and perform various other administrative tasks within a Kubernetes environment. It supports a wide range of commands and options to interact with the Kubernetes API server and control the desired state of your cluster.

These are the key components of GKE that work together to enable the efficient deployment and management of containerized applications using Kubernetes. GKE abstracts away all the complexities of managing the underlying infrastructure, allowing developers to focus on building and deploying their applications. Now that we understand GKE architecture at a high level, it is imperative to understand Google's shared responsibility model.

GKE shared responsibility model

Google is responsible for the following aspects of GKE security:

- The hardware, firmware, kernel, OS, storage, network, and other components of the underlying infrastructure. This includes encrypting data at rest by default, encrypting data in transit, custom-designed hardware, private network connections, physical access protection for data centers, and secure software development methods.
- The OS for the nodes, such as **Container-Optimized OS** (**COS**) or Ubuntu. GKE makes any changes to these images available as soon as possible. These images are automatically installed if you have auto-upgrade enabled. This is the foundation of your container; it is different from the operating system that runs inside it.
- Kubernetes is a container orchestration system. GKE provides the most recent Kubernetes upstream versions and supports numerous minor versions. It is Google's responsibility to provide updates to these, including patches.
- The control plane, which comprises the master VMs, the API server, and other components that run on those VMs, as well as the etcd database, is managed by GKE. This includes upgrades and patches, scalability, and repairs, all of which are backed up by a **Service-Level Agreement** (**SLA**).
- IAM, Cloud Audit Logging, Cloud Logging, Cloud Key Management Service, Cloud Security Command Center, and other Google Cloud integrations. These controls are also accessible for IaaS workloads on GKE in Google Cloud.

Now let us see what Google Cloud customers are responsible for:

- The nodes on which your workloads are executed. Any additional software installed on the nodes, as well as any configuration changes made to the default, are your responsibility.
- Customers are also in charge of keeping their nodes up to date. Google automatically supplies hardened VM images and configurations, maintains the containers required to run GKE, and gives OS patches—all they need to do is upgrade. If they use node auto-upgrade, Google will upgrade these nodes for them.
- The workloads themselves, which include your application code, Dockerfiles, container images, data, RBAC/IAM rules, and running containers and pods. To help safeguard your containers, you can use GKE security features and other Google Cloud products such as **Container Threat Detection** (**KTD**), part of Security Command Center. Sometimes Google customers also use third-party products such as Twistlock for runtime protections.

Now that you understand GKE architecture and the shared responsibility model, let us see what are the challenges for container security.

Container security

Container security is about making sure that a container-based system or workload is protected by using different security methods and technologies. This includes securing the container image during creation, securing the deployment of the image, and making sure that the container environment is secure during runtime.

Let us look at threats and risks in containers.

Threats and risks in containers

NIST defines cybersecurity risks as relating to the loss of confidentiality, integrity, and availability of information and data and its potential adverse impact on an organization or business. The NIST definition of container security risks includes the following:

- **Image risks**: Image configuration defects, embedded malware, embedded clear text secrets, the use of untrusted images
- **Registry risks**: Insecure registry connections, stale images, and inadequate authentication and authorization to the registry are common risks
- **Orchestrator risks**: Unrestricted administrative access, unauthorized access, poorly separated inter-container network traffic, and the mixing of workload sensitivity levels
- **Container risks**: Vulnerabilities within the runtime software, unbounded network access from containers, insecure container runtime configurations, app vulnerabilities, and rogue containers
- **Host OS risks**: Large attack surface, shared kernel, vulnerabilities in host OS components, unauthorized user access permissions, manipulation with the host OS file system, and unauthorized hosts joining the cluster and running containers

Considering these risks, at a high level, the container threats in the cloud can be separated into three categories:

- **Container infrastructure security**: Is your infrastructure secure for deploying containers? The threats in this category include the following:
    - Privilege escalation
    - Credential compromise
    - Kubernetes API compromise
    - Overprivileged users

- **Software supply chain threats**: Is your container image secure to build and deploy? The threats in this category include the following:
  - Unpatched vulnerability
  - Supply chain vulnerability
  - Exploiting a common library
- **Runtime security**: Is your container secure to run? The threats in this category include the following:
  - DDoS
  - Node compromise and exploitation
  - Container escape

Now let us look at the GKE security features to understand how GKE protects container workloads from these threats.

GKE security features

The contents of your container image, the container runtime, the cluster network, and access to the cluster API server all play a role in protecting workloads in GKE. Let us understand a few security features in GKE.

Namespaces

In Kubernetes, namespaces are used to separate groups of resources in a cluster. Resources within a namespace must have unique names, but this requirement doesn't apply across namespaces. It's important to note that namespace-based scoping only applies to resources that are specific to a namespace, such as Deployments and Services, and doesn't apply to objects that are used across the entire cluster, such as Nodes, StorageClass, and PersistentVolume.

Namespaces in Kubernetes are intended for situations where there are multiple users spread across different teams or projects. If your cluster only has a small number of users, you may not need to worry about namespaces.

Namespaces allow you to group resources together by providing a unique scope for their names. Each resource can only be in one namespace, and namespaces cannot be nested inside each other. They also enable multiple users to share cluster resources through resource quotas.

If you only need to identify slightly different resources, such as different versions of the same software, you don't necessarily need to use multiple namespaces. Instead, you can use labels to differentiate resources within the same namespace.

Let us look at how GKE provides access control.

Access control

GKE supports two options for managing access to resources within the Google Cloud project and its clusters:

- **Kubernetes Role-Based Access Control** (**RBAC**)
- **Identity and Access Management** (**IAM**)

These two methods are similar in function, but they are focused on different types of resources. Let us look at them now.

Kubernetes RBAC

This is a built-in feature of Kubernetes that grants granular permissions to Kubernetes cluster objects. Within the cluster, permissions are represented by ClusterRole or Role objects. Roles in Kubernetes RBAC provide permissions at the namespace level, while ClusterRoles offer cluster-wide permissions. Proper usage of Roles and ClusterRoles is crucial for maintaining security by adhering to the principle of least privilege and enabling the separation of duties. Regular review and auditing of assigned Roles and ClusterRoles are essential to ensure that permissions align with security requirements. RoleBinding objects allow Kubernetes users, Google Cloud users, Google Cloud service accounts, and Google groups to have access to roles.

Kubernetes RBAC is the ideal choice if you primarily utilize GKE and need fine-grained rights for every item and operation in your cluster. GKE has built-in support for RBAC that allows you to create fine-grained Roles that exist within the GKE cluster. A role can be scoped to a specific Kubernetes object or a type of Kubernetes object and defines which actions (called verbs) the role grants in relation to that object. A **RoleBinding** is also a Kubernetes object, which grants roles to users. In GKE, the subject can be any of the following principals:

- Cloud Identity user
- Google Cloud IAM service account
- Kubernetes service account
- Cloud Identity Google group

Now, let us understand IAM for GKE.

IAM

IAM oversees Google Cloud resources, such as clusters and the types of things that can be accessed within them. GKE permissions/roles are assigned to IAM principals.

Within IAM, there is no way to provide permissions to individual Kubernetes objects. For example, you can give a user permission to create **Custom Resource Definitions** (**CRDs**), but you cannot give them permission to create only one CustomResourceDefinition or to confine creation to a specific namespace or project cluster. If an IAM role is applied at the folder level, it grants rights to all clusters in the project or all clusters in all child projects.

IAM is a suitable choice if you use different Google Cloud components and do not need to maintain detailed Kubernetes-specific rights.

IAM and Kubernetes RBAC collaborate to help you manage cluster access. RBAC regulates access at the cluster and namespace levels, whereas IAM manages access at the project level. To work with resources in your cluster, an entity must have proper rights at both levels.

Now let us see how GKE manages secrets.

Secrets

Secrets are Kubernetes objects that store sensitive data in your clusters, such as passwords, OAuth tokens, and SSH keys. Secrets are more secure than plaintext ConfigMaps or Pod specs for storing sensitive data. Secrets provide you control over how sensitive data is utilized and decrease the chance of unauthorized persons accessing it. You can create a secret using the kubectl CLI or a YAML file.

Now let us understand how GKE tracks changes and activity.

Auditing

According to the Kubernetes documentation, Kubernetes auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster. The cluster audits the activities generated by users, by applications that use the Kubernetes API, and by the control plane itself.

Auditing allows cluster administrators to answer the following questions:

- What happened?
- When did it happen?
- Who initiated it?
- Where was it observed?
- From where was it initiated?
- To where was it going?

Audit records begin their lifecycle inside the `kube-apiserver` component. Each request on each stage of its execution generates an audit event, which is then pre-processed according to a certain policy and written to a backend. The policy determines what's recorded and the backends persist the records. The current backend implementations include log files and webhooks.

The Kubernetes documentation goes on to say that each request can be recorded with an associated stage. The defined stages are as follows:

- `RequestReceived`: The stage for events generated as soon as the audit handler receives the request, and before it is delegated down the handler chain.
- `ResponseStarted`: Once the response headers are sent, but before the response body is sent. This stage is only generated for long-running requests (for example, watch).
- `ResponseComplete`: The response body has been completed and no more bytes will be sent.
- `Panic`: Events generated when a panic occurred.

The Kubernetes audit policy defines rules about what events should be recorded and what data they should include. GKE receives the log entries from the Kubernetes API server, and it applies its own policy to determine which entries get written to the project's admin logs. For the most part, GKE applies the following rules to log entries that come from the Kubernetes API server:

- Entries that represent `create`, `delete`, and `update` requests go to the Admin Activity log
- Entries that represent `get`, `list`, and `updateStatus` requests go to the Data Access log

Admin Activity logs are turned on by default in your Google Cloud project. They can't be turned off; however, you must turn on Data Access logs. Let us move on to understanding the features of GKE logging.

Logging

By default, GKE clusters are natively integrated with Cloud Logging (and Monitoring). When you create a GKE cluster, both Monitoring and Cloud Logging are enabled by default. That means you get a monitoring dashboard specifically tailored for Kubernetes and your logs are sent to Cloud Logging's dedicated, persistent datastore and indexed for both searches and visualization in the Cloud Logs Viewer.

There are several ways to access your logs in Cloud Logging depending on your use case.

You can access your logs using the following:

- **Cloud Logging console**: You can see your logs directly from the Cloud Logging console by using the appropriate logging filters to select Kubernetes resources such as cluster, node, namespace, pod, or container logs. Here are some sample Kubernetes-related queries to help get you started.

- **GKE console**: In the **Kubernetes Engine** section of the Google Cloud console, select the Kubernetes resources listed in **Workloads**, and then the **Container** or **Audit Logs** link.
- **Monitoring console**: In the **Kubernetes Engine** section of the Monitoring console, select the appropriate cluster, nodes, pods, or containers to view the associated logs.
- **gcloud command line tool**: Using the `gcloud logging read` command, select the appropriate cluster, node, pod, and container logs.

These are all the methods of logging that you should be conversant with. Here are some best practices for containerized applications when it comes to logging:

- Use the native logging mechanisms of containers to write the logs to `stdout` and `stderr`.
- If your application cannot be easily configured to write logs to `stdout` and `stderr`, you can use a sidecar pattern for logging.
- Log directly with structured logging with different fields. You can then search your logs more effectively based on those fields.
- Use severities for better filtering and reducing noise. By default, logs written to the standard output are on the `INFO` level and logs written to the standard error are on the `ERROR` level. Structured logs with a JSON payload can include a severity field, which defines the log's severity.
- Use the links to the logs directly from the **Kubernetes Engine** section of the Cloud console for containers, which makes it quick to find the logs corresponding to the container.

Now that we have auditing and logging out of the way, let us move on to understand how GKE does network isolation.

Network Policies

You use Kubernetes NetworkPolicies for certain applications in your cluster if you wish to manage traffic flow at the IP address or port level (OSI layer 3 or 4). **Network policies** are an application-centric construct that allows you to declare how a pod can communicate across the network with various network *entities* (we use the term *entity* here to avoid overusing terms like *endpoints* and *services*, which have unique Kubernetes implications). Other connections are not affected by NetworkPolicies since they have a pod on one or both ends.

To manage communication between your cluster's pods and services, you can utilize GKE's network policy enforcement. Pod-level firewall rules are created using the Kubernetes Network Policy API to construct a network policy. These firewall rules control which pods and services in your cluster can communicate with one another.

When your cluster is hosting a multi-level application, defining a network policy allows you to enable things such as defense in depth. For example, you can set up a network policy to prevent a compromised frontend service from communicating directly with a billing or accounting server many tiers down.

A network policy can also help your application host data from several users at the same time. You can, for example, define a tenant-per-namespace paradigm to provide secure multi-tenancy. Network policy rules can ensure that pods and services in one namespace cannot access pods and services in another namespace in such a model.

Another form of network isolation is by using a feature called GKE private clusters, whereby you can host a private cluster, that is, a cluster with no public IP address. Let us understand this.

GKE private clusters

In private clusters, nodes only have internal IP addresses to isolate nodes from inbound and outbound connectivity to the internet.

Worker nodes in a private cluster only have internal IP addresses. The control plane, on the other hand, has both a private and a public destination. When creating a private cluster, you can disable access to the public endpoint. An internal load balancer provides access to the private endpoint. If you wish to reach the control plane from a GCP region other than where it is deployed, you will need to allow global access. Let us look at the architecture of private clusters now:

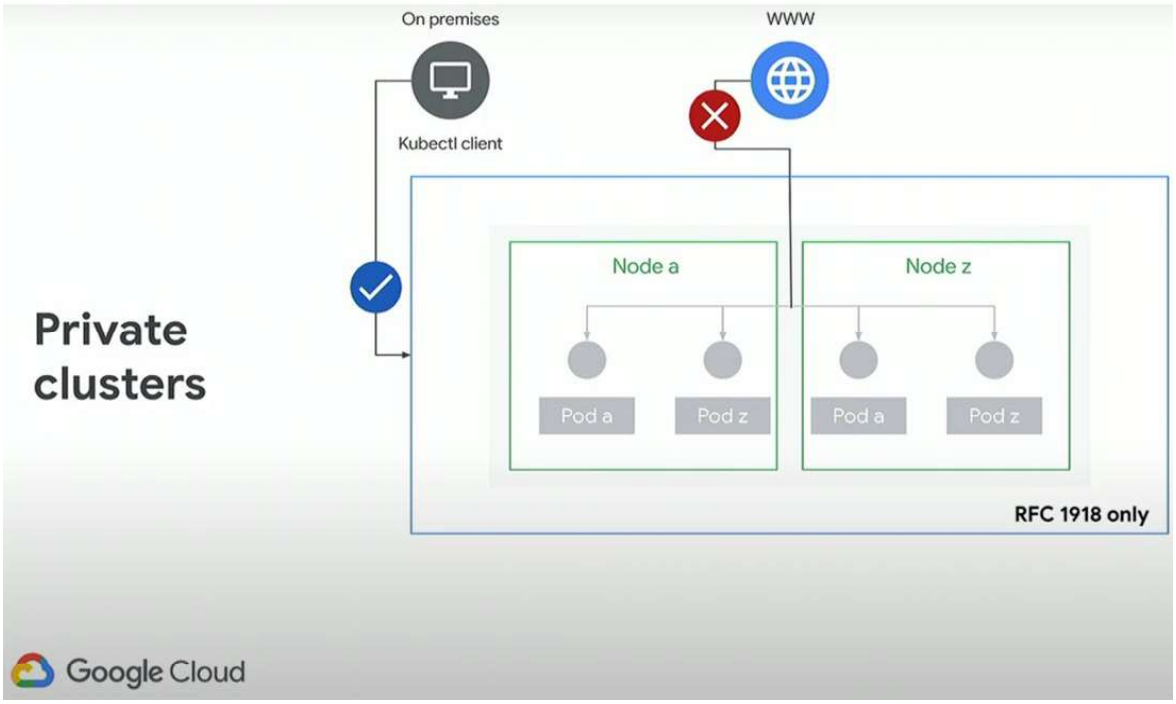

Figure 15.4 – Private clusters

As seen in *Figure 15.4*, using a private cluster has the additional security benefit that nodes are not exposed to the internet. As with public clusters, you can also use GKE's authorized networks feature with private clusters to restrict access to the master API.

A service mesh is an increasingly popular method of implementing network policies for GKE. We will understand that now.

Service mesh

A service mesh is a Layer 7 proxy. Microservices can use this service mesh to abstract the network. This effectively abstracts how inter-process and service-to-service communications are handled in the K8S cluster. In Kubernetes, the service mesh is usually implemented as a set of network proxies. These proxies, which are deployed as a *sidecar* of an application, serve as an entry point for service mesh functionality and manage communication between microservices. Istio is a popular open service mesh that connects, manages, and secures microservices in a consistent manner. It allows you to manage traffic flows between services, enforce access controls, and aggregate telemetry data without modifying the microservice code. The following are the benefits that Istio offers:

- Automatic load balancing for HTTP, gRPC, WebSocket, MongoDB, and TCP traffic
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection
- A configurable policy layer and API that supports access controls, rate limits, and quotas
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress
- Secure service-to-service communication in a cluster with strong identity-based authentication and authorization
- You configure Istio access control, routing rules, and so on by using a custom Kubernetes API, either via kubectl or the Istio command-line tool istioctl, which provides extra validation

Google recommends using Anthos Service Mesh, Google's full-supported distribution of Istio.

When you create or update a cluster with Istio on GKE, the following core Istio components are installed:

- The *istiod* control plane, which provides the following:
  - Traffic management
  - Security
  - Observability
- The Istio ingress gateway, which provides an ingress point for traffic from outside the cluster

The installation also lets you add the Istio sidecar proxy to your service workloads, allowing them to communicate with the control plane and join the Istio mesh.

So far, we have understood the GKE security features. Now, we will look at how to secure a container image, which can be a source of vulnerability.

Container image security

Container images are *baked* by a pipeline, a series of steps that add the required components on top of each other. The application is deployed on the very top, as a last step. The following figure shows the process of a container pipeline published by NIST.

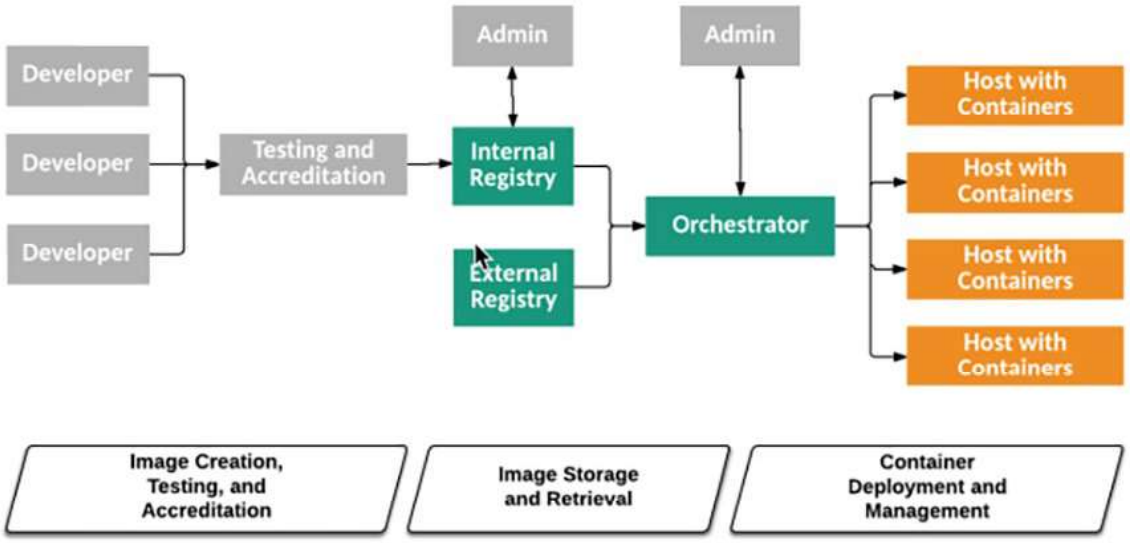

Figure 15.5 – Container pipeline

In *Figure 15.5*, as recommended by NIST (`https://packt.link/luNhB`), the container pipeline follows a controlled process for image generation:

1. Developer systems generate images and send them for testing and accreditation.
2. Testing and accreditation systems validate and verify the contents of images, sign images, and send images to the registry.
3. Registries store images and distribute images to the orchestrator upon request.
4. Orchestrators convert images into containers and deploy containers to hosts.
5. Hosts run and stop containers as directed by the orchestrator.

Now that you understand what the pipeline does, here are some best practices for container image security as recommended by Google:

- Use Google-maintained base images
- Do not rely on packages in third-party repositories (if possible, host an internal repository of the packages that are scanned for vulnerability)
- Do not include secrets in images

- Scan images for vulnerabilities
- Stop the deployment of the container if the images do not pass attestations (see the *Binary Authorization* section later in the chapter)

Now that you understand the best practices for image security, let us see how to scan an image for vulnerabilities on Google Cloud.

Container vulnerability scanning on Google Cloud

Container scanning is a service on Google Cloud that identifies known vulnerabilities in Debian, Ubuntu, Alpine, Red Hat, and CentOS packages inside the container on GCP. This feature can be enabled in Artifact Registry.

There are two types of scans supported by container scanning:

- **On-demand scanning**: Using the gcloud CLI, you may scan container images locally on your PC or in your registry on demand. This gives you the freedom to tailor your CI/CD pipeline to your specific needs, for example, when you need to obtain vulnerability results.
- **Automated scanning**: Container Analysis searches container images in Artifact Registry and Container Registry for vulnerabilities and keeps the vulnerability information up to date. Scanning and continuous analysis are the two major jobs in this technique.

When new images are uploaded to Artifact Registry or Container Registry, Container Analysis examines them. This scan extracts information on the container's system packages. Based on the image's digest, the images are only scanned once. This means that adding or changing tags will not result in new scans; only changing the image's content would. Here is what happens when you enable scanning:

- When you upload an image, Container Analysis creates occurrences for vulnerabilities found. It continually checks the information for scanned images in Artifact Registry and Container Registry for additional vulnerabilities after the initial scan.
- Container Analysis changes the metadata of the scanned images as new and updated vulnerability information is received from vulnerability sources, establishing new vulnerability occurrences for fresh images, and eliminating vulnerability occurrences that are no longer valid.
- Container Analysis only updates vulnerability metadata for images pushed or pulled in the previous 30 days. Vulnerability metadata older than 30 days is archived by Container Analysis. To re-scan an image with archived vulnerability metadata, push or pull that image.