WHIZLABS

Google Cloud Certified Professional Data Engineer

Cheat Sheet

Quick Bytes for you before the exam!

WHIZLABS

Index

—-Back to Index—

—-Back to Index—

Designing Data Processing

Google Cloud Architecture Framework: Security, Privacy, and Compliance

The **Google Cloud Architecture Framework** provides best practices across six key pillars: system design, operational excellence, security, reliability, cost, and performance optimization. A core principle, **defense in depth**, ensures multiple layers of protection between threats and assets. The **security, privacy, and compliance** pillar emphasizes scalable, default-enabled security to safeguard data, ensure privacy, and meet regulatory requirements.

Key Features & Best Practices – Security, Privacy, and Compliance

Cloud Architecture Framework are as follows:

🔐 Shared Responsibility & Shared Fate

- Google secures infrastructure; users manage their data and access.
- Shared Fate promotes active collaboration for improved security outcomes.

🛡️ Security Principles

- Implement **defense-in-depth** across all layers.
- Design **secure, decoupled systems** and document security needs.
- **Automate** sensitive task deployments and security monitoring.
- Meet regional **compliance** and **data residency** requirements.
- **Shift security left**: Integrate security early in the development lifecycle.

⚠️ Risk & Asset Management

- Identify and catalog risks using industry frameworks.
- Apply safeguards: contracts, tech controls, and third-party validations.
- Use cloud tools for **asset management**, automate tracking, and monitor for compliance.

👥 Identity & Access Management

- Use a single identity provider, enable **SSO** and **MFA**.
- Enforce **least privilege**, audit access, and automate policy controls.
- Plan service account usage and restrict resource access.

🖥️ Compute & Container Security

- Use hardened VM images and **Confidential Computing**.
- Disable external IPs unless required, monitor and update instances.

🌐 Network Security

- Deploy **Zero Trust networks** and Web Application Firewalls.
- Monitor traffic and disable default networks where possible.

🔒 Data Security

- Classify and encrypt data, control access and location.
- Use **Secret Manager**, manage metadata, and monitor usage.

—-Back to Index—

🚀 Secure Application Deployment

- Automate releases, scan for vulnerabilities, and monitor application code.
- Encrypt container images and control data movement across perimeters.

📋 Compliance & Sovereignty

- Use **Assured Workloads**, compliance blueprints, and automation.
- Manage **data, operational, and software sovereignty**.

🕵️ Privacy & Threat Monitoring

- Lock down confidential data, monitor for phishing.
- Extend Zero Trust to hybrid workforces.
- Monitor networks, detect threats, and prevent data exfiltration.

Use Case :

- **Security & Compliance:** Use the framework to automate compliance, perform vulnerability scans, and apply security blueprints for secure deployments.
- **Data Security:** Leverage encryption, access controls, and secret management to protect sensitive data effectively.

Exam Tips:

- Understand best practices for IAM, especially the principle of least privilege.
- Familiarize yourself with encryption techniques at rest and in transit. Understand different key management options like Customer-managed encryption keys and customer-supplied encryption keys.
- Understand the risk assessment framework to identify risks and techniques for mitigating the risk.
- Understand important concepts for Network security like WebAppliation Firewall, VPC Service Control, and Zero Trust Security Model/Network.
- Understand the security core principles like layered security,defense-in-depth approaches.

Database Migration Service - Overview

What is Database Migration?

Database Migration involves transferring schema, data, and metadata from a source to a destination.

Google Database Migration Service is a serverless, managed tool that enables secure, low-downtime migration to Google Cloud, offering scalability, high availability, and ease of use without infrastructure overhead.

Features: **Database Migrations Service supports the following migrations:**

🔄 Homogeneous Migration

Migrate similar database engines (e.g., MySQL → Cloud SQL for MySQL).
Uses native replication tools for MySQL/PostgreSQL.

🔁 Heterogeneous Migration

Migrate across different engines (e.g., Oracle → Cloud SQL for PostgreSQL).
Uses CDC-based replication for continuous sync.

📊 Migration Management

Tracks migration status, performs initial snapshots, and supports continuous replication.
Simplifies networking and connectivity setup.

Other Key Characteristics:

🧩 Ease of Use

Simple UI to create, configure, and monitor migration jobs.
Pre-migration validation ensures a smooth experience.

🤖 AI-Assisted Conversion

Gemini helps convert stored procedures, triggers, and functions to PostgreSQL dialect.
Provides side-by-side code comparison with explanations.

☁️ Serverless & Secure

No server provisioning needed.
Secure, private connectivity protects data in transit.

✅ Best Practices

- Assess DB size and environment
- Plan thoroughly with timeline and validation steps
- Monitor destination for performance and issues
- Ensure adequate network bandwidth
- Backup destination DB post-migration

🚀 Use Case

Migrate on-prem MySQL/PostgreSQL to Cloud SQL or AlloyDB.
Modernize Oracle workloads to PostgreSQL-based services for scalability and performance.

—-Back to Index—

Exam Tips:

- Examine Database Migration Service documentation and understand the homogeneous and heterogeneous migration and replication techniques.
- Make a high-level understanding of supported source and destination databases especially which databases are supported for continuous migration.
- Understand how migration jobs are monitored and performance tuning.
- Practice answering scenario-based questions that ask you to pinpoint possible data migration performance improvement or job failure reasons.
- Learn about the permission needed in the source database to facilitate Data Migration.
- Learn about the different connectivity methods for Database Migration Services like- IP allowlist, forward SSH tunnel, and VPC peering network connectivity methods.

Migrate to Google Cloud: Best practices for validating a migration plan

Google Cloud migration is the process of transferring applications, data, infrastructure, security, and other objects from On-Prem Infrastructure or other cloud service providers to the Google Cloud Platform. To minimize risk and maximize benefits, organizations should follow best practices and prepare a comprehensive migration plan for a seamless transition to Google Cloud.

🛠️ Migration Strategies

- **Rehost:** Lift & shift with minimal changes
- **Replatform:** Optimize after migration
- **Refactor:** Re-engineer for cloud-native use
- **Re-architect:** Modernize for scalability
- **Rebuild:** Replace existing with new cloud-native apps
- **Repurchase:** Move to SaaS-based solutions

🚀 Google Cloud Adoption Framework

Helps identify key tasks and goals to streamline your cloud journey.

📋 Migration Phases

- **Assess:** Inventory apps, dependencies, TCO, and benchmarks
- **Plan:** Set up infrastructure, networking, IAM, and migration strategy
- **Deploy:** Execute automated or manual app deployment
- **Optimize:** Monitor usage, improve performance, reduce cost, plan for DR

✅ Best Practices

- Inventory and assess app downtime, failure modes, clustering, and performance
- Right-size resources and network capacity
- Plan phased rollouts with a rollback strategy
- Monitor logs, metrics, and enable alerts for key thresholds

💡 Use Cases

- Move on-prem apps to GCP for better performance
- Migrate to managed databases (Cloud SQL, AlloyDB)
- Speed up app development and reduce downtime
- Improve cost and infrastructure efficiency through modernization

Exam Tips:

- Familiarize yourself with Google Cloud migration tools like Database Migration Services and Transfer Appliance.
- Understand key migration phases and strategies.Know security controls and measures.
- Learn optimization techniques for resources and cost.

—-Back to Index—

Google Cloud Architecture Framework: Reliability

The Google Cloud Architecture Framework provides best practices for designing secure, efficient, and cost-effective cloud systems. **Reliability**, one of its six pillars, emphasizes four principles:

1. Prioritize reliability over new features.
2. Users define reliability through satisfaction with service performance.
3. Aim for enough reliability to satisfy customers, not 100% at excessive cost.
4. Balance innovation and reliability using error budgets.

🔧 Key Approaches for High Reliability:

Measure Reliability:

- **SLI** (Service Level Indicator): Measures user satisfaction.
- **SLO** (Service Level Objective): The target value for SLI. The service is considered reliable when SLI is at or above this threshold.
- **SLA** (Service Level Agreement): A formal contract with users that outlines what happens if SLOs are missed.
- **Error Budget**: Represents how much downtime is allowed. It is calculated as 100% minus the SLO, showing the permissible level of failure over time.

Scale & Availability:

- **Redundancy**: Building redundant systems ensures high availability even during failures.
- **Multi-Zone & Multi-Region Architecture**: Ensure that your system is resilient to failures in specific zones or regions by deploying across multiple locations.
- **Disaster Recovery**: Always have a recovery plan in place, including data replication across regions.
- **Degrade Gracefully**: In case of overload, the system should degrade service without causing major disruptions.

Operational Processes:

- **Progressive Rollouts**: Gradually introduce changes using **canary testing** to reduce risks.
- **Automation**: Automate build, test, and deployment to improve reliability and speed.
- **Testing & Recovery**: Regularly test failure recovery and disaster recovery plans to ensure reliability in case of emergencies.

Alerts:

- **Early Detection**: Set up alerts that can detect issues early before they affect users.
- **Outlier Values**: Alert on outliers rather than averages to catch unexpected issues.
- **Symptom-Based Alerts**: Instead of alerting on specific causes, focus on symptoms that show the system may be failing.

—-Back to Index—

📊 Reliability for Data Engineering:

- **Durability**: Data products should ensure that data remains intact, uncorrupted, and recoverable during failure events.
- **High Availability**: Data should be accessible, even if some components fail. This ensures users always have access to the data.
- **Data Consistency**: Ensure that data across all components is consistent, so the data retrieved matches what was written or modified.
- **Recovery**: Plan for data recovery in case of accidental deletion or pipeline failure. Always have a backup strategy.
- **Disaster Planning**: Set clear **RTO** (Recovery Time Objective) and **RPO** (Recovery Point Objective) metrics for how long systems can be down and how much data loss is acceptable.
- **Autoscaling**: Automatically adjust computational resources based on demand, ensuring that performance stays optimal without manual intervention.
- **Graceful Decommissioning**: For services like **Dataproc**, decommissioning worker nodes should not impact the ongoing jobs or services.

✅ Best Practices:

- **Backup & Recovery**: Implement a robust backup strategy and regularly test it to ensure data can be recovered.
- **Least Privilege Access Control**: Minimize the risk of unauthorized access or accidental failures by applying the principle of least privilege in your system's permissions.
- **Materialized Views in BigQuery**: Use materialized views to speed up queries and reduce costs, especially when querying large datasets.
- **Monitor SLIs & SLOs**: Keep track of service performance indicators and ensure your systems meet the required objectives.
- **Cross-Region Storage**: Utilize cross-region storage solutions to improve disaster recovery and ensure data redundancy across multiple locations.
- **Autoscaling**: Enable autoscaling to manage high traffic and adjust resources based on usage demand.
- **CI/CD**: Set up **CI/CD** pipelines to automate testing, deployment, and version control for your data pipelines.
- **Security**: Ensure strong security practices, including encryption, access controls, and regular audits to protect data integrity and confidentiality.

🏥 Use Case - Healthcare:

- **Patient Data Reliability**: Healthcare providers use the Google Cloud Reliability Framework to ensure that patient data is always accessible, secure, and protected from loss. With features like autoscaling and rapid failure detection, healthcare services can maintain continuity even during unexpected demand surges.

Exam Tips:

- Understand the design and operation principles of Reliability.
- Familiarize yourself with Key Metrics for Reliability such as SLI, SLO, SLA, and Error Budgets.
- Understand key concepts like high availability, scalability, durability, disaster recovery, and replication.
- Study best practices implemented for reliability against each data product.
- Recognize the cloud data products' high-level disaster recovery architecture and how metrics like recovery time objectives (RTO) and recovery point goals (RPO) fit with business requirements.
- Familiarize yourself with Synchronous and asynchronous data replication strategies and data backup procedures.
- Understand how to automate failover and recovery processes and read more about Terraform on automating resource provisioning.

WHIZLABS

Getting Started with Migrating to Google Cloud

🚀 Google Cloud Migration Overview

Migration to Google Cloud involves moving workloads from on-prem, private, or other clouds to GCP to gain cost-efficiency, better security, flexibility, and speed.

📍 Migration Phases

1. **Define Starting Point:**
 Identify source environment – On-prem, Private Cloud, or other CSPs.
2. **Define Workload Type:**
    - ***Legacy:*** Hard to modify and maintain.
    - ***Cloud-Optimized:*** Stable, portable, and secure.
3. **Choose Migration Strategy:**
    - 🔄 Rehost – Lift & shif
    - ⚙️ Replatform – Lift & optimize
    - 🛠️ Refactor – Move & improve
    - 🧱 Re-architect – Modernize

      🔄 Rebuild – Replace entirely
    - 🛒 Repurchase – Switch to SaaS
4. **Assess Cloud Readiness:**
 Use Google Cloud Adoption Framework.
5. **Define Migration Path:**
 Assess → Plan → Deploy → Optimize

🔍 Assessment Phase

- Inventory workloads and dependencies
- Train teams and run PoCs
- Estimate TCO using GCP Pricing Calculator
- Pick tools and strategy
- Finalize plan and validate timeline

🏗️ Planning & Foundation

- **Resource Hierarchy:** Set structure, billing, and policies
- **IAM:** Define roles, access, audit policies
- **Security:** Encrypt data, enforce policies
- **Monitoring & Logging:** Plan observability stack
- **Governance:** Ensure compliance and maintainability

🚢 Deployment Approaches

- Manual
- Using CM tools
- Containers/Kubernetes
- Automated CI/CD pipelines

♻️ Optimization

- Define optimization goals
- Use autoscaling
- Automate monitoring
- Tune based on usage
- Use Infrastructure as Code (IaC)

🧠 Best Practices

- Full workload assessment (downtime, redundancy, clustering)
- Rollback plans & staged rollouts
- Alert stakeholders
- Remove PoCs from production
- Safely retire old environments
- Migrate data before apps

📦 Use Cases

- Data Migrations – M&A, consolidation
- Application Migrations – Better scale, CX, cost savings

Exam Tips

- Know migration types & use cases
- Understand GCP Adoption Framework
- Know tools like Transfer Appliance
- Learn to estimate TCO
- Master IAM & Resource Hierarchy
- Understand optimization and IaC

—-Back to Index—

Data governance in BigQuery

Data governance ensures your data is private, accurate, accessible, and useful across its lifecycle. In BigQuery, it supports security, cost control, risk mitigation, and compliance.

🛠️ Key Features

1. Access Control

- **IAM Roles**: Manage access to BigQuery projects, datasets, tables, and views.
- **Column- & Row-Level Access**: Restrict access to sensitive data at a granular level.
- **VPC Service Controls**: Restrict data transfer across boundaries.

2. Audit Logging

- Track user actions and system events to support policy enforcement.

3. Data Stewardship

- Classify, mask, redact, or encrypt sensitive data.
- **Data Masking**: Built on column-level access for obfuscating sensitive info.

4. Encryption

- Data is encrypted at rest and in transit by default, with optional custom settings.

5. Metadata Management

- Use **Data Catalog** to tag and organize resources for search and classification.

6. Data Quality

- Powered by **Dataplex** to manage accuracy and consistency.
- **Data Lineage**: Track data flow and history.
- **Profile Scans**: Analyze patterns (nulls, uniqueness, etc.).
- **Quality Scans**: Identify and fix data quality issues.

✅ Best Practices

- Use IAM roles + column/row-level controls.
- Classify/tag data by sensitivity and apply policy tags.
- Enable audit logs to monitor access and actions.
- Use **Data Catalog** for tagging and organization.
- Enable **Data Lineage API** in Dataplex to track data flow.

📦 Use Cases

- **Healthcare**: Meet regulations, classify data, track lineage, protect sensitive info.
- **Finance**: Control access, manage quality, audit data movement, and catalog resources.

Exam Tips

- Know IAM roles (predefined & custom).
- Data masking: only at **column level**.
- Use **policy tags** and **Cloud DLP** for classification.
- Understand **Data Catalog** usage and tag management.
- Learn how **Dataplex** supports lineage, profiles, and quality scans.

—-Back to Index—

Data Ingestion & Processing

Planning and Building a Streaming Data Pipeline on Google Cloud

A **streaming data pipeline** processes and transfers real-time data from source to destination in three stages:

1. **Ingestion**
2. **Processing**
3. **Storage**

Unlike batch processing, streaming enables near real-time insights.

⚙️ **Key Components**

📥 Ingestion – *via Pub/Sub*

- Fully managed messaging service
- Publishers send messages; subscribers consume them

🧮 **Processing – *via Cloud Run / Dataflow***

- **Cloud Run**: Lightweight processing using stateless containers
- **Dataflow**:
  - Apache Beam-based; supports **exactly-once** processing
  - Ideal for windowing, filtering, joins, aggregation
  - Uses **PCollections** & **PTransforms**
  - Autoscaling enabled
  - Offers built-in templates and Vertex AI integration

🗄️ **Storage**

- Streamed data stored in BigQuery, Cloud Storage, or Bigtable

📊 **Planning Considerations**

- Define **SLOs** (data freshness, latency, correctness)
- Choose **single vs. multiple pipelines** based on SLO priority
- Account for **data source/sink scalability** and **network topology**
- **Regionalize** pipelines near source/sinks for efficiency
- Ensure **security & compliance** with CMEK, Cloud HSM, private IPs

Best Practices

- 📈 Monitor job status, latency, data freshness
- 🪟 Use **windowing & triggers** for chunking & emitting results
- 🧱 Use **Dataflow templates** and shared logic libraries

- 🔁 Deduplicate messages for exactly-once processing
- 🧪 Keep logic simple & testable
- 🔐 Ensure service accounts have correct permissions
- 📦 Choose optimized formats like **Avro**
- 🧵 Plan separate pipelines for standard vs. high-priority data

💡 Use Cases

- Clickstream Analytics
- IoT Sensor Monitoring
- Social Media Sentiment Analysis
- Ride-hailing Matching Systems

Exam Tips

- Understand **windowing, triggering, sinks**, and **exactly-once** processing
- Know monitoring metrics (latency, parallelism, CPU/memory)
- Learn optimization techniques (autoscaling, shuffling)
- ⚠️ Important: Understand **"Drain Job"** – gracefully shuts down while processing buffered data

—-Back to Index—

Cloud Deploy: Delivery Pipelines and Targets

🚀 What is Google Cloud Deploy?

A **fully managed CI/CD service** that automates app delivery to environments like **GKE, Cloud Run, and Anthos** using a predefined pipeline. It supports **centralized control**, **one-click promotion**, and **rollback**.

⚙️ Key Features

- **Delivery Pipeline**: YAML-defined workflow that outlines deployment sequence and targets.
- **Targets**: Represents environments like GKE, Cloud Run; supports **parallel deployment**.
- **Releases:** Snapshot of pipeline + config (Skaffold + Manifest) for every code/config change.
- **Automation**: Auto-promote and auto-advance rollouts using Skaffold for rendering + deployment.
- **Pipeline Management**: View, update, suspend, delete pipelines; configure alerts.

✅ Best Practices

- 📈 Use **Cloud Monitoring** & **Cloud Logging**
- 🔐 Implement **Scoped IAM Roles**
- 📦 Maintain Infra as Code with **Version Control**
- ⚖️ Use **Canary Deployment** for safer rollouts
- ⚙️ Leverage **Skaffold** for flexible manifest rendering
- 🔄 Enable **Auto-scaling & Resource Optimization**

💡 Use Cases

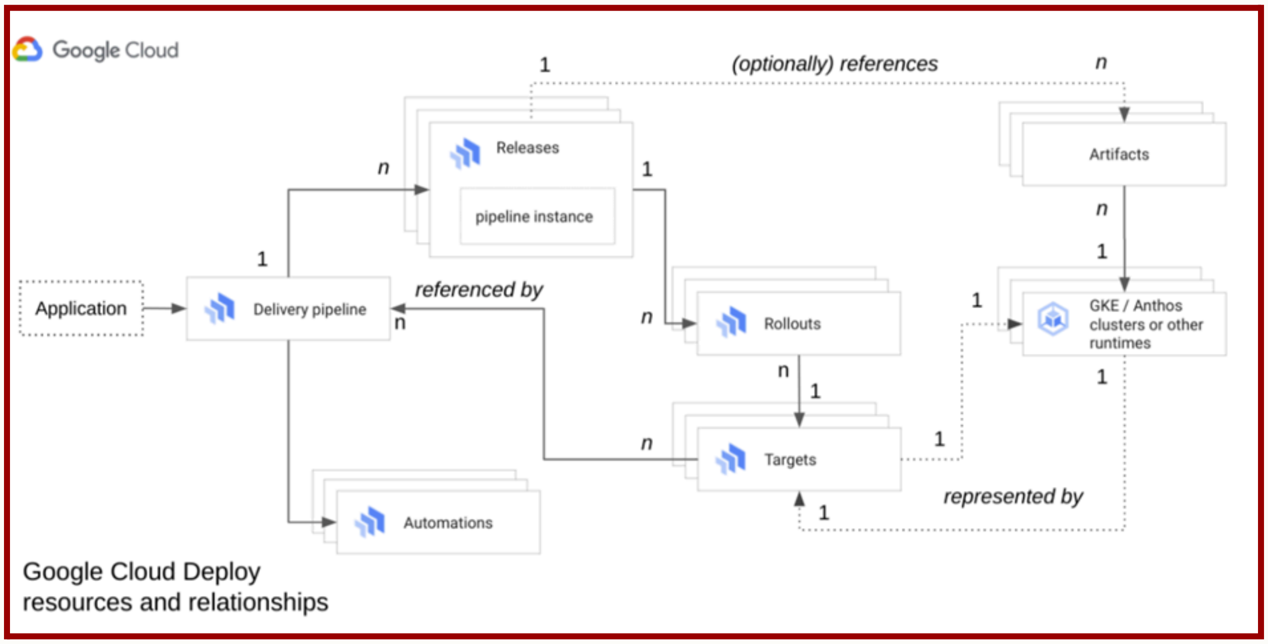
- Canary Deployment for safer user exposure
- Parallel deployment across multiple environments
- Automated app delivery to live routes
- Quick rollbacks for failed releases

Exam Tips

- Know concepts: **Pipeline, Release, Targets, Manifest, Rendering**
- Understand **YAML & Skaffold** configs
- Learn automation triggers & rollout rules
- Be familiar with **metrics, monitoring, and logging**
- Recognize supported **deployment environments**

—-Back to Index—

This is the visual representation of the Cloud Deploy Resources:



Cloud Scheduler – Overview

⏰ What is Google Cloud Scheduler?

A **fully managed cron job service** for running scheduled tasks to trigger jobs or automate infrastructure operations. It ensures **at least once delivery**, supports **retry policies**, and integrates with **HTTP/S, Pub/Sub, and App Engine**.

⚙️ Key Features

- Fully managed, no infra maintenance
- Configurable retry policies with backoff
- Supports HTTP/S, Pub/Sub, App Engine targets
- At-least-once reliable delivery
- Integrated with Cloud Logging & IAM
- Cron-style flexible scheduling
- Simple pay-per-use pricing

✅ Best Practices

- 🧩 Manage all jobs in a single interface
- 🕒 Use correct cron format & timezone
- 🔁 Configure retries and error handling
- 🔍 Monitor via Cloud Logging & Audit Logs
- 📜 Document schedule, targets, retries
- 🛠️ Use Terraform for consistency & version control

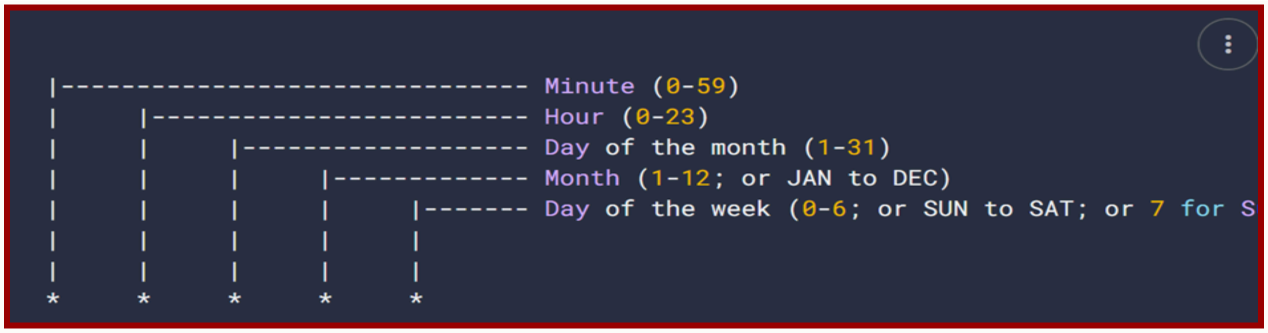
💡 Use Cases

- Cloud Infrastructure Automation
- Recurring Batch & Big Data Job Scheduling
- ML Model Retraining on New Data

Exam Tips

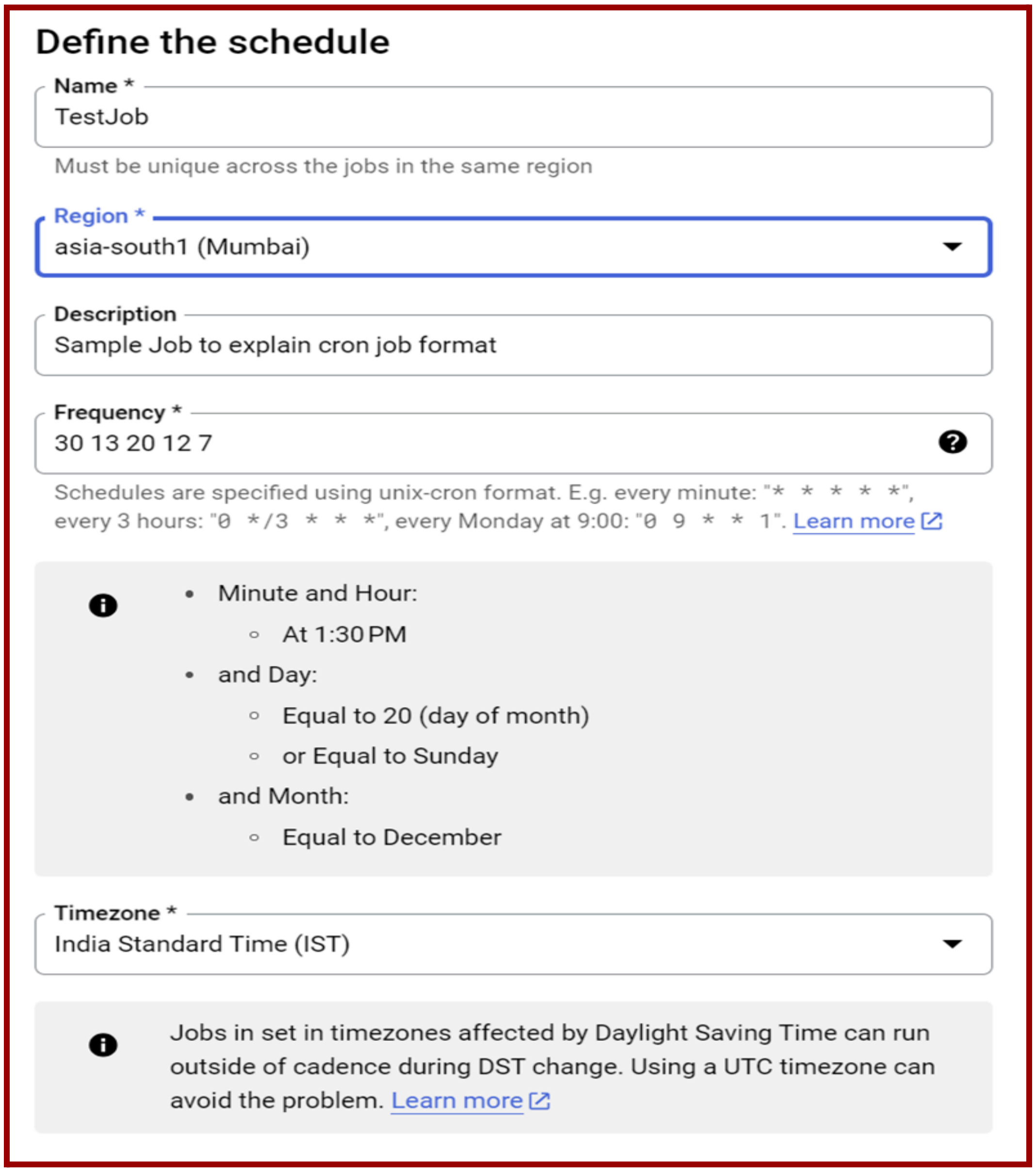
- Understand cron syntax & timezone handling
- Learn retry/backoff configuration
- Know target types: HTTP, Pub/Sub, App Engine
- Explore IAM roles for job-level access control

—-Back to Index—



```
|------------------------------- Minute (0-59)
|     |------------------------- Hour (0-23)
|     |     |------------------- Day of the month (1-31)
|     |     |     |------------- Month (1-12; or JAN to DEC)
|     |     |     |     |------- Day of the week (0-6; or SUN to SAT; or 7 for S
|     |     |     |     |
|     |     |     |     |
*     *     *     *     *
```

Here is an example of the sample job schedule which signifies the job is to be run on every 20th and every Sunday in December.



Define the schedule

Name *
TestJob
Must be unique across the jobs in the same region

Region *
asia-south1 (Mumbai)

Description
Sample Job to explain cron job format

Frequency *
30 13 20 12 7

Schedules are specified using unix-cron format. E.g. every minute: "* * * * *", every 3 hours: "0 */3 * * *", every Monday at 9:00: "0 9 * * 1". Learn more

- Minute and Hour:
  - At 1:30 PM
- and Day:
  - Equal to 20 (day of month)
  - or Equal to Sunday
- and Month:
  - Equal to December

Timezone *
India Standard Time (IST)

Jobs in set in timezones affected by Daylight Saving Time can run outside of cadence during DST change. Using a UTC timezone can avoid the problem. Learn more

Orchestration in Google Cloud

🔗 What is Orchestration?

Orchestration connects multiple tasks/services into a unified workflow to streamline execution, automate processes, and optimize performance. Google Cloud offers various tools tailored to different orchestration needs.

⚙️ Key Tools & Features

Feature / Tool	Cloud Scheduler	Workflows	Cloud Composer
Type	Cron-based scheduler	Serverless workflow orchestration	Apache Airflow-based orchestration
Use Case	Simple, recurring tasks	Real-time orchestration of multiple HTTP-based services	Complex ETL/ELT and batch data workflows
Execution Format	Cron expression	YAML / JSON with step-by-step execution	Python DAGs
Scalability	Limited to single service	Scalable and stateful	Scales for batch jobs, not low-latency
Retry Support	Yes	Yes	Yes
Integration Support	GCP services (HTTP/HTTPS, Pub/Sub, App Engine)	Google Cloud, private APIs, 3rd-party APIs	BigQuery, Dataflow, Dataproc, hybrid/multi-cloud
Best For	Scheduling standalone jobs	Low-latency, real-time workflows involving multiple services	Data pipelines with complex dependencies

✅ Best Practices

- 🎯 Choose tools based on use case (Scheduler → single, Workflows → HTTP services, Composer → data/ETL). Avoid hardcoding URLs, use parameterization
- 🗂️ Simplify DAGs to reduce dependencies
- 🕒 Use correct cron format with timezone in Scheduler
- 🛠️ Use Terraform for versioned orchestration setup
- 🔐 Assign dedicated service accounts & fine-grained IAM
- 📊 Monitor with Cloud Logging & Audit Logs

💡 Use Cases

- ETL/ELT Data Pipelines (Cloud Composer)
- Scheduled Cron Jobs (Cloud Scheduler)
- HTTP-based Multi-Service Orchestration (Workflows)
- Automated Cloud Infra Operations (Workflows)
- Batch & Real-Time Data Pipelines

Exam Tips

- Know the difference: Composer vs. Workflows
- Understand DAG setup and CI/CD deployment. Pick the right tool per use case

Master cron syntax & retry settings. Learn logging & troubleshooting for orchestration tools.

—-Back to Index—

Workload management using Reservations

💡 Billing Models

- **On-Demand:** Pay per byte processed at query time
- **Flat-Rate:** Reserve slots monthly or annually for predictable workloads
- **Flex Slots:** Short-term (60-sec min) slot reservations, ideal for testing or burst workloads

⚙️ Key Concepts

- **Slots:** Virtual CPUs for executing BigQuery jobs
- **Commitment:** Reserved slot capacity for a specific time period
- **Reservation:** Allocation of slots across projects or workloads
- **Assignment:** Maps projects/folders/orgs to reservations
- **Idle Slots:** Preemptible slot usage from other reservations (default enabled)
- **Regional Restriction:** Reservations can't span across regions

💲 Slot Pricing Options

Pricing Model	Description	Use Case
On-Demand	No commitments, pay-per-use	Ad-hoc, unpredictable workloads
Flat-Rate	Monthly/annual commitments for fixed slots	Steady, predictable usage
Flex	Short-term slot bursts (min 60s)	Testing, debugging, short-term spikes

✅ Best Practices

- **Mix pricing models** for flexibility and cost control
- Run workloads for **7+ days** before using the slot estimator
- Assign resources at the **project/org level** for better slot management
- **Disable idle sharing** for critical jobs if needed
- Create **high/low-priority reservations** for job importance
- Use **performance and cost recommendations** for autoscaling & planning

💡 Use Cases

- **Critical Business Jobs**: Reserve most slots for priority processing
- **Multi-Department Orgs**: Department-level reservations + central billing
- **Short-Term Needs**: Use Flex slots for spikes, testing

Exam Tips

- Know all **slot pricing options**: On-Demand, Flat-Rate, Flex
- Understand **reservations, assignments, slots, commitments**
- Remember **reservations are regional,** Learn **resource hierarchy** and its impact on slot assignment
- Understand **idle slot behavior** & how to manage it.
- Use tools like **INFORMATION_SCHEMA, Cloud Logging, Jobs API** to monitor slot usage

—-Back to Index—

Cloud Composer

Managed Apache Airflow for orchestrating data pipelines across hybrid and multi-cloud.
Seamless integration with BigQuery, Dataflow, Dataproc, Cloud Storage, Pub/Sub.
Ideal for batch workloads with slight task execution delays.

⚙️ Key Concepts

- **Fully Managed Airflow**: Scalable Apache Airflow service on GCP
- **Python-based DAGs**: Define workflows as code using Python
- **Environment Components**: GKE cluster, Airflow scheduler, triggerer, workers, web server, DB, Cloud Storage bucket
- **Monitoring**: Built-in dashboard, Cloud Logging & Monitoring integration
- **Auto-Scaling**: Dynamically adjusts worker count based on demand

✅ Best Practices

- Spread out DAG schedules to avoid resource contention
- Minimize task dependencies in DAGs
- Use **Pub/Sub** for DAG-to-DAG communication
- Set dagrun_timeout to manage job durations
- Use **Terraform** for infra-as-code deployments
- Assign tailored IAM roles to Composer environments
- Monitor **Environment Health, Resource Usage, Task Metrics**
- Limit job retries to prevent overload

💡 Use Cases

- Cross-cloud Data Pipelines
- ETL/ELT Workflow Orchestration
- ML Pipeline Scheduling
- Financial Batch Processing
- Healthcare Data Workflows

Exam Tips

- Know how to configure and schedule **Airflow DAGs**
- Understand **Cloud Composer environment architecture**
- Learn how to **monitor and scale** Composer environments
- Familiarize with Composer's **GCP service integrations**
- Use **CI/CD pipelines** for DAG deployment

—-Back to Index—

Cloud Data Storage Options

Cloud Storage - Overview and Integration with Google Cloud services and tools

What It Is:

Scalable, durable, and secure object storage for unstructured data. Access from anywhere via HTTP/HTTPS with 11 9's durability.

💡 **Integration Use Cases:**

- **BigQuery** – Run large-scale queries directly on GCS data.
- **DataProc / Dataflow** – Process data at scale with open-source & serverless tools.
- **Vertex AI** – Store ML model inputs/outputs.
- **Cloud Functions** – Trigger events on object changes.
- **Looker Studio** – Build dashboards from stored data.
- **Firebase** – Upload files from mobile/web apps.
- **App Engine / CDN / Load Balancer** – Host and serve static assets.
- **Pub/Sub** – Event notifications on object updates.
- **Compute Engine** – Boot VMs from images stored in GCS.
- **Cloud SQL** – Import/export data.
- **Security** – Sensitive Data Protection, VPC Service Controls.
- **Monitoring** – Cloud Logging, Error Reporting.
- **Data Transfer** – Move data in/out with Storage Transfer Service.

✅ **Best Practices:**

- Choose right storage class (Standard, Nearline, etc.)
- Use lifecycle policies for cost efficiency.
- Enable Cloud CDN for edge caching.
- Track object changes with Cloud Functions.
- Apply encryption and access controls.
- Monitor usage via Cloud Logging/Monitoring.

Exam Tips:

- Understand storage classes, lifecycle management, and access control (Signed URLs).
- Know integrations: Cloud Function, Pub/Sub, App Engine, etc.
- Familiarize with encryption options: default, CMEK, CSEK.

—-Back to Index—

Optimizing Costs in Google Cloud Storage

What is Google Cloud Storage?

Cloud Storage is a managed, scalable object storage service for unstructured data with no set capacity limits (though individual object size limits apply). Pricing depends on storage class, data processing, and network usage. Cost optimization is driven by access patterns, retention needs, and performance requirements.

🔍 Key Factors:

- Storage Class:
  - Choose based on **access frequency**:
    - Standard: For frequently accessed data.
    - Nearline: For data accessed ~once a month.
    - Coldline: For data accessed ~once a quarter.
    - Archive: For rarely accessed data (once/year).
  - Lower access = lower storage cost, **but** higher **retrieval fees**.
- Location Type:
  - Multi-regional and Dual-regional offer **higher availability**, but cost more.
  - Regional is cheaper and good for localized access or backups.
- Data Access & Retrieval Fees:
  - Retrieval fees **apply** to Nearline, Coldline, and Archive.
  - No retrieval fees for Standard.
- Operations:
  - Actions like listing, copying, or modifying objects incur **operation charges**.
- Data Transfer:
  - **Outbound data (egress)** costs money.
  - Internal transfers (within the same region) are usually free or low-cost.

✅ Best Practices:

- Lifecycle Management:
  - Automatically **move** data to lower-cost classes or **delete** after a certain time.
- Monitoring & Budgets:
  - Use **Cloud Monitoring** and **Cloud Billing Alerts** to keep track of usage and costs.
- Avoid Object Versioning:
  - Each version counts towards total storage—only enable if needed.
- Data Compression & Delta Transfers:
  - Compress files to reduce storage.
  - Transfer only changes (delta) instead of full files.
- Tag Resources: Use labels/tags for **cost allocation** and tracking across teams or projects.

—-Back to Index—

💡 **Use Cases:**

- Hosting static assets: **media**, **web images**, **music**, etc.
- Long-term **log storage** or **backup archives**.
- Storage for **analytics datasets**, **ML model inputs/outputs**.

📘 **Exam Tips:**

- Know when to use each **storage class** (based on access frequency + retention needs).
- Understand **cost-saving tools**: lifecycle policies, storage class tiers, monitoring.
- Familiarize with **data transfer options**:
    - **Online** (Storage Transfer Service)
    - **Offline** (Transfer Appliance).
- Encryption and **access controls** (IAM, CMEK, CSEK) affect cost and security posture.

Data Lake Overview [Creating, Cost Controls and Monitoring]

A data lake is a centralised repository designed to store, process, and secure large volumes of data. It can store any format (structured, semi-structured, unstructured) of data in its original format and process it regardless of any size restriction. Organizations should consider the following characteristics when building a data lake: Data Ingestion, Data Organization and Transformation, Data Discovery, Data Quality, and interoperatibility.
Centralized storage of structured, semi-structured, and unstructured data for analytics, AI, and reporting.

🔧 Components:

- **GCS**: Store raw data in any format.
- **BigQuery**: Query structured data from GCS.
- **Dataflow / Dataproc**: Process/transform data.
- **Data Catalog**: Discover, organize metadata.

💸 Cost Control:

- Use **appropriate storage classes** + **lifecycle policies**.
- **Compress** data, optimize **BigQuery queries** (partition/cluster).
- Use **materialized views** for frequent access.
- Scale Dataflow/Dataproc based on demand.

📊 Monitoring:

- Enable **Cloud Monitoring & Logging**.
- Use **IAM** to manage access.
- Track **job performance, usage, and costs**.

✅ Best Practices:

- Classify & catalog data early.
- Set **access control policies** (least privilege).
- Implement **security, governance, lineage** tracking.

💡 Use Cases:

- Recommendation engines (e.g. video streaming)
- Risk models (e.g. financial services)

Exam Tips:

- Know data classification, lifecycle, access control (Uniform vs Fine-grained).
- Be clear on BigQuery/GCS integration and cost-saving techniques.

—-Back to Index—

Data lake: Configuring Data discovery, Access Management, and Encryption

What is a Data Lake?

A data lake is a centralized repository for storing, processing, and securing large volumes of structured and unstructured data in its original format. Cloud Storage and BigQuery work together to support analytics and querying. Key management areas include data discovery (cataloging, exploration), access control (IAM), and encryption (default, CMEK, CSEK) to ensure security and governance.

Discovery:

- Use **Data Catalog** for metadata, lineage, and search.
- Enable **Dataplex** for automatic discovery and classification.

Access Management:

- Control via **IAM roles** and policies at bucket/object level.

Encryption:

- **Default encryption** (AES-256), **TLS** in transit.
- **CMEK**: Customer-managed keys.
- **CSEK**: Customer-supplied keys.

✅ Best Practices:

- Plan access/security based on **data classification**.
- Monitor access logs via **Cloud Audit Logs**.
- Rotate keys regularly; apply **TLS + AES-256** encryption.
- Enforce **least privilege** access.

💡 Use Cases:

- Telcos using churn models from historical user data.
- Secure enterprise-level analytics with fine-grained controls.

Exam Tips:

- Know IAM access levels (Uniform vs Fine-grained).
- Be confident in encryption strategies (default, CMEK, CSEK).
- Understand Dataplex, Data Catalog, key management.

—-Back to Index—

High availability and replicas | Memorystore for Redis Cluster

What is GCP Memorystore for Redis?

It is a fully managed service powered by Redis in-memory data store to build a highly available and scalable application cache that offers sub-millisecond data access. Memorystore is based on open-source Redis version 7.2 or earlier and compatible with Redis Cluster, Redis (built-in persistence capability), and Memcached(does not have built-in persistence). GCP Memorystore enables applications to achieve extreme performance by providing ultra-fast, real-time data access. Fully managed in-memory data store for ultra-fast, sub-millisecond data access. Based on Redis v7.2 or earlier, supports Redis, Redis Cluster, and Memcached.

⚙️ Tiers

- **Basic**: Standalone, no failover, data lost on failure.
- **Standard**: High availability with read replicas and auto-failover. Suitable for critical apps.

🌟 Key Features

- **High Availability**: Replication & auto-failover across zones.
- **Scalable**: Up to 300 GB (Redis) or 250 nodes (Cluster).
- **Monitoring**: Metrics via Cloud Monitoring (CPU, memory, latency).
- **Security**: Encryption in transit & at rest, IAM roles, audit logging.

✅ Best Practices

- **CPU Utilization**: Avoid expensive commands; scale or shard as needed.
- **Memory Management**: Monitor memory ratios; set maxmemory conservatively.
- **Connection Handling**: Use retry logic, timeouts, connection pooling.
- **Backups**: Use Redis RDB snapshots.
- **Maintenance**: Set maintenance windows to manage downtime.

💡 Use Cases

- **Caching**: Web sessions, API responses.
- **Gaming**: Leaderboards, fast player profile access.
- **Streaming**: Real-time feeds (IoT, social).

📝 Exam Tips

- Understand tiers: Redis vs Redis Cluster vs Memcached.
- Learn failover, replication, backup options.
- Review performance metrics and best practices.

—-Back to Index—

Google Cloud Data Warehouse and Memorystore Overview

BigQuery is a fully managed, serverless, petabyte scaled low-cost data warehouse service for analytics. BigQuery enables you to focus on analyzing data to discover meaningful insights while utilizing familiar SQL and built-in machine learning at an unrivaled price performance. Memorystore is a fully managed in-memory data store that provides sub-millisecond data access, scalability, and high availability for many applications. It offers three main options for in-memory data storage:

- Memorystore for Redis
- Memorystore for Redis Cluster
- Memorystore for Memcached
- **BigQuery**: Serverless data warehouse; supports SQL & ML; separates compute & storage.
- **Memorystore:** Fast, scalable in-memory cache; supports Redis & Memcached.

💡 BigQuery Use Cases

- Fraud Detection
- Product Recommendations
- Demand Forecasting

💡 Memorystore Use Cases

- Gaming Leaderboards
- Geo Queries
- Session Storage
- Token Caching

Exam Tips (BigQuery + Memorystore)

- Know Redis vs Redis Cluster features.
- Familiarize with BigQuery's ML capabilities, partitioning, and query optimizations.
- Understand how IAM and audit logs apply to both.

—-Back to Index—

Data lake modernization solutions

Data Lake Modernization is the process of enhancing and strengthening existing data lake infrastructure to make it more secure, scalable, fast, and accessible through leveraging GCP services. Google Cloud's data lake solution can host and analyze any type of data including structured, unstructured, and semi-structured data. It supports low-cost data ingestion, storage, and analysis of a large volume of diverse, full-fidelity data. It can facilitate re-hosting the data lake even without rebuilding your on-premises data lake, autoscaling a resource-intensive data workload. Modernize legacy data lakes using GCP tools to improve performance, scalability, and cost-efficiency.

🧱 Core Components

- **Cloud Storage**: Stores all data types.
- **BigQuery**: Runs analytics & queries.
- **Dataproc**: Spark/Hadoop jobs for lift & shift.
- **Dataflow**: ETL & stream/batch processing.
- **AI/ML Tools**: AI Platform Notebooks, Spark + GPUs.

🔐 Governance & Security

- Metadata & Lineage: Use Data Catalog.
- IAM for access control, DLP for sensitive data.
- Cloud Monitoring & Logging for observability.

💡 Use Cases

- Hadoop migration
- Financial data integration
- Retail analytics pipelines
- ML model training & serving

Exam Tips

- Know core GCP services: BigQuery, Dataproc, Dataflow, Pub/Sub.
- Understand decoupling compute from storage.
- Learn governance: metadata, catalog, DLP.
- Monitor with Cloud Monitoring & Logging.
- Use autoscaling and lifecycle policies to control cost.

BigQuery

Google Cloud BigQuery

BigQuery is a serverless, AI-ready, fully managed, scalable, multi-engine, multi-format, multi-cloud data analytics platform that helps you extract the most value from your data. Serverless, scalable, AI-ready analytics platform that processes TBs in seconds and PBs in minutes. Supports multi-cloud, multi-engine, multi-format queries.

⚙️ Key Features

- **Serverless Architecture**: Auto-scales storage & compute independently.
- **High Performance**: Distributed engine handles real-time analytics.
- **GoogleSQL**: Default SQL dialect (based on SQL:2011).
- **ML with BigQuery ML**: Train/test models using SQL—no data movement.
- **Vertex AI Integration**: Pull/push data between BigQuery & Vertex AI.
- **Gemini AI**: Gen AI tools assist with query writing & insights.
- **BI Engine**: Fast dashboarding with no performance compromise.
- **Analytics Hub**: Secure inter-org data sharing via pub/sub model.

📊 Monitoring & Optimization

- **Cloud Monitoring**: Tracks slot usage, scanned bytes, execution time.
- **Cloud Logging**: Alerting on job failures, errors.
- **Audit Logs**: Track data access and user actions.
- **Dry Run & Query Validator**: Estimate cost/performance pre-run.
- **INFORMATION_SCHEMA Views**: Query metadata & job stats.

🛠️ Best Practices

- Use **IAM roles** for fine-grained access.
- Apply **Partitioning & Clustering** to improve performance & reduce cost.
- Apply **Data Governance**: Classification, cataloging, lineage.
- Use **Query Validator** to catch inefficient queries early.
- Monitor **model performance** using correct ML metrics.

💡 Use Cases

- Sales Forecasting via BQ ML
- Realtime Dashboards with BI Tools
- Recommendation Engines
- GeoSpatial Data Analysis
- Cross-org Data Sharing via Analytics Hub

Exam Tips

- Know Gemini, Analytics Hub, BI Engine. Understand ML model types in BigQuery.

Understand serverless design—decoupled compute/storage. Familiarize with monitoring tools: Logging, Audit Logs, Schema Views. Query tuning: partitioning, clustering, dry run, caching.

—-Back to Index—

BigQuery Views and Types of Views

- Its views are virtual tables defined by SQL queries. The view can be created on top of tables and other views and gives a logical representation of data from these sources.
- Views do not support insert, update, or delete statements; they are read-only and do not store any data.
- By providing a restricted view of a limited dataset or subset, views are a useful tool for managing access to the underlying database.

🧱 Types of Views

- **Logical View:**
  - No data stored
  - Reflects live changes
  - Lower cost, slower query
- **Materialized View:**
  - Stores precomputed results
  - Periodic refresh (default: every 30 min)
  - Faster, costlier (due to storage)

🧰 Best Practices

- Use views to restrict data access.
- Refresh/recreate views after schema changes.
- Use **Materialized Views** for frequently run, compute-heavy queries.
- Avoid nesting Materialized Views.
- Set **expiration dates** for cleanup.

💡 Use Cases

- Subset access for analysts
- Fast reporting dashboards
- Aggregated data reuse
- Complex joins across massive data

Exam Tips

- Logical vs Materialized: cost, latency, refresh, storage.
- Views = Read-only; no insert/update/delete.
- INFORMATION_SCHEMA contains view metadata.
- Understand performance trade-offs.

BigQuery: Analyse data with Looker Studio and Interactive or Batch query jobs

Looker Studio is a no-cost self-service data visualization tool that allows you to create and consume dashboards and customizable reports from various data sources. BigQuery's Interactive and Batch Query features integrate well with Looker Studio to provide scalable performance in building dashboards even when the BigQuery dataset expands.

📊 BigQuery + Looker Studio Integration

Use Looker Studio (free) to visualize BigQuery data via dashboards and reports.

⚙️ Key Features

- Connect 800+ data sources.
- Build & share interactive dashboards.
- Supports real-time and batch data.
- Template library + embeddable reports.
- BI Engine, Materialized Views, and Caching support.

🔄 Query Modes

- **Interactive Queries**:
  - Triggered by user interaction
  - Real-time, but more costly
  - Limited concurrency
- **Batch Queries**:
  - Scheduled, cheaper
  - Ideal for recurring reports

🛠️ Best Practices

- Pre-aggregate data before reporting.
- Apply filters, aggregates, and caching.
- Monitor query stats via INFORMATION_SCHEMA.JOBS.
- Schedule batch queries for large datasets.
- Authorize connections & share datasets securely.

💡 Use Cases

- Monthly/Quarterly Sales Dashboards
- Healthcare Analytics (clinical trials, patient outcomes)
- Real-time KPI Monitoring
- Departmental Self-Service BI

Exam Tips

- Know how to connect Looker Studio to BigQuery.
- Understand Interactive vs Batch query trade-offs.
- Learn performance features: BI Engine, caching, window functions.

Operations and Management

Cloud IAM

Cloud Identity and Access Management (IAM) allows users to manage access control for specific Google Cloud resources and helps prevent access to other resources. IAM follows the security principle of least privilege which grants users access permission necessary for their task. IAM works by defining who (identity) has what access (role) for which resource. IAM offers a consolidated view of the security policies and controls permission and enhances security and compliance.

🧱 Core Components

- **Principal:** User Account, Service Account
- **Role:** Basic, Predefined, Custom
- **Policies:** Allow, Deny
- **Resources:** Organization, Folder, Project, Resources

🔐 Governance & Security

- Centralized Access Control
- Principle of Least Privilege
- Audit Logs
- MFA, Just-in-time Privileged Access

💡 Use Cases

- Manage Service Accounts
- Centralized Access Control
- Workload Identity Federation
- Security and Compliance

Exam Tips

- Least Privilege, Service Account, RBAC
- Allow/Deny Policies, Resource Hierarchy
- Security & Compliance, Audit Logs
- Identity Federation, Just-in-Time Access

—-Back to Index—

Google Cloud Observability - Overview

Google Cloud Observability assists organizations in understanding the behaviour, health, and performance of the deployed applications and infrastructure. Google Cloud Observability is a collection of services integrated to collect, analyze, and correlate telemetry data such as metrics, logs, traces, and other data generated by the application and infrastructure. The deep insights obtained through this exercise enable organizations to swiftly identify issues, determine root causes, and execute remedial measures.

🔑 **Key Features:**

- **Cloud Logging:** Managed log monitoring and analysis.
- **Cloud Monitoring:** Metrics, events, dashboards, and alerts.
- **Cloud Trace:** Distributed tracing for latency and performance insights.
- **Cloud Profiler:** CPU and memory profiling.
- **Error Reporting:** Centralized error management and analysis.

💡 **Best Practices:**

- Enable audit logs.
- Aggregate logs centrally.
- Set proper log retention policies.
- Correlate logs, metrics, and traces.
- Configure alerts based on priorities.
- Estimate and track costs of logging/monitoring.

📝 **Use Cases:**

- Application/Infrastructure Health Monitoring
- Security Monitoring
- Multi-cloud and On-prem Monitoring

Exam Tips:

- Understand the best practices and eliminate the obvious wrong answers.
- Familiarize yourself with key features like Cloud Monitoring, Cloud Logging, Cloud Profiler, Cloud Trace, and Error Reporting.
- Be aware that the Cloud Debugging service has been deprecated.
- Learn the logical flow in Cloud Observability (collect logs, monitor, generate reports, and create alerts).

—-Back to Index—

Cloud Key Management Service Overview

What is a Key Management Service (KMS)?

- Key Management Service (KMS) service enables users to conduct cryptographic operations in a single, centralized cloud service as well as create, import, and manage cryptographic keys.
- It provides scalable and secure cloud key management of symmetric(used for data encryption) and asymmetric(used for digital signature to ensure data integrity) cryptographic keys.
- It supports high-performance key operations for large volumes of data effortlessly and satisfies the privacy, compliance, and regulatory needs of the organization.

🔑 Key Features:

- Manage encryption keys (symmetric/asymmetric).
- Centralized key storage, rotation, and destruction.
- Cloud Hardware Security Module (HSM) integration.
- Customer-managed encryption keys (CMEK).
- External Key Manager (EKM) support.

💡 Best Practices:

- Rotate keys regularly.
- Enable audit logging for key usage.
- Use IAM roles to control key access.
- Backup/recovery plan for key loss.

📝 Use Cases:

- Regulatory Compliance (e.g., HIPAA, PCI DSS)
- Secure Key Management (CMEK, BYOK)

Exam Tips:

- Familiarize yourself with KMS key lifecycle management and key rotation best practices.
- Understand how keys are access-controlled, monitored, and audited.
- Be aware of different encryption key types (symmetric/asymmetric) and how they are used in KMS.
- Learn about the KMS key hierarchy and the concept of key locations (regional, multi-regional, and global).

—-Back to Index—

Google Cloud Encryption – Overview

Google employs encryption to transform readable data known as plaintext into an unreadable format (called as ciphertext) to prevent unauthorized access. This process ensures that only authorized persons with decryption keys can convert ciphertext back to the original form. Google Cloud uses a robust encryption method to safeguard customer data while in transit and at rest. Additionally, Google offers users the option to manage their own encryption keys to provide an extra layer of protection.

Key Features:

- **Encryption at Rest:** AES-256, Key Encryption Keys (KEK), and envelope encryption.
- **Customer-managed encryption keys (CMEK):** Extra layer of control.
- **Encryption in Transit:** TLS, ALTS, AES GCM for data in transit.

💡 **Best Practices:**

- Use KMS for key management.
- Regularly rotate encryption keys.
- Use encryption across all layers (network, application, storage).
- Use IAM for key access control.

📝 **Use Cases:**

  - Protecting Sensitive Financial/Healthcare Data
  - Encrypting Customer Data in Compliance with Industry Regulations

Exam Tips:

- Understand encryption at rest and in transit, and how data is encrypted using DEKs and KEKs.
- Familiarize yourself with KMS and CMEK for better control over encryption keys.
- Learn security protocols used for encryption in transit like TLS, ALTS, and AES GCM.
- Study how encryption ensures data authenticity, integrity, and privacy during transit.

—-Back to Index—

Monitoring and Troubleshooting Processes in GCP

Monitoring and troubleshooting are essential for the success of Google Cloud Platform (GCP) data engineering. Resource and cost optimization can be accomplished by gathering monitoring metrics, and visualizing the metrics. Early issue detection can be achieved through setting up alerting policies and searching error logs available in Cloud Log. The monitoring tools in GCP provide thorough insights into databases, asynchronous communication tools, data pipelines, and jobs, and they enable pre-emptive problem identification and timely troubleshooting when errors occur.

Key Features:

- **Dataflow:** Metrics, job monitoring, failure logs.
- **BigQuery:** Query metrics, data scans, execution time.
- **DataProc:** Cluster and job monitoring.
- **Pub/Sub:** Message throughput, delivery latency monitoring.
- **Cloud Storage:** Metrics for errors, data transfer, and requests.

💡 **Best Practices:**

- Treat data as a product; establish data quality and lineage.
- Evaluate data sources and processing tools.
- Use Cloud Monitoring and logging for pipeline visibility.
- Apply data governance practices (access controls, lifecycle management).

📝 **Use Cases:**

- Real-Time Data Analytics (e.g., Sentiment Analysis, Market Sentiment)
- Data Pipeline Monitoring and Optimization

Exam Tips:

- Understand BigQuery architecture and query optimization for cost management.
- Familiarize yourself with Dataflow and DataProc, and their respective use cases.
- Know how to monitor Pub/Sub for message throughput and latency issues.
- Apply best practices in data pipeline design and monitoring with Cloud Monitoring.
- Understand Cloud Storage capabilities like security, encryption, and lifecycle policies.

Architecting disaster recovery for cloud infrastructure outages

Disaster Recovery(DR) is the systematic procedure by which an organization prepares to respond to any kind of disaster event, whether it's man-made or natural, by restoring access and operational capabilities to its IT infrastructure. Architecting and designing the Cloud Infrastructure with Google Cloud products having built-in DR mechanisms can make the platform more resilient and minimize outages.

🔑 **Key Features:**

- **Zones/Regions:** Multi-regional resources for higher reliability.
- **Replication Strategies:** Synchronous for zero RPO, asynchronous for higher RPO.
- **Automated Failover:** Hot/warm recovery based on RTO.

💡 **Best Practices:**

- Distribute resources across multiple zones/regions.
- Automate infrastructure provisioning with Terraform.
- Frequent testing of disaster recovery plans.
- Monitor recovery with Google Cloud Observability.

📝 **Use Cases:**

- Financial Institutions and E-commerce Platforms needing zero downtime.
- Critical Application Recovery with minimal RPO/RTO.

Exam Tips:

- Understand key DR metrics like RTO, RPO, SLA, and SLO.
- Learn about DR strategies such as Hot, Warm, and Cold recovery.
- Familiarize yourself with data replication strategies (synchronous vs. asynchronous).
- Study how to automate failover and recovery processes using Terraform.
- Be aware of Google Cloud's products for DR (Compute Engine, Cloud Storage, Load Balancing, etc.).

—-Back to Index—

Preparing and Using Data for Analysis

🔑 **Key Features:**

- **Connect to BI Tools:**
  - Integrate data sources (e.g., databases, warehouses) with BI platforms like Looker, Tableau, Power BI.
  - Use connectors/APIs for efficient data flow.
  - Weigh performance trade-offs between direct connections and data extracts.
- **Materialized Views & Precalculated Fields:**
  - Use materialized views to store results of complex queries for faster performance.
  - Define precalculated fields for reusable metrics and derived dimensions.
- **Define Time Granularity:**
  - Choose suitable aggregation (daily, monthly, etc.).
  - Handle time zones consistently for accurate analysis.
- **Troubleshoot Query Performance:**
  - Use query plans to diagnose slow queries.
  - Optimize with indexing, partitioning, and caching.
- **IAM & Cloud DLP:**
  - Control access using IAM roles.
  - Use Cloud DLP to protect sensitive information and maintain compliance.

💡 **Best Practices:**

- Use materialized views for frequently queried data.
- Precalculate metrics where appropriate to reduce BI load time.
- Align time zones across datasets.
- Use IAM for fine-grained access control.
- Apply Cloud DLP to mask or tokenize PII.

📝 **Use Cases:**

  - Performance-optimized Dashboards for Executives
  - Cross-functional Access to Governed Reports
  - Time-series Analysis in Sales or Marketing

Exam Tips:

- Know the difference between materialized views and calculated fields in BI tools.
- Understand how time granularity affects analysis and forecasting accuracy.
- Be aware of common query performance issues (e.g., full table scans, missing indexes).
- Expect questions on IAM roles for securing access to dashboards and datasets.

—-Back to Index—

🔁 Sharing Data

🔑 **Key Features:**

- **Define Data Sharing Rules:**
    - Set permissions and policies to govern who can view or access data.
    - Apply access control at dataset and report level.
- **Publish Datasets and Reports:**
    - Curate and share reliable datasets for reuse.
    - Ensure datasets are documented and regularly refreshed.
- **Use Analytics Hub:**
    - Central platform for discovering and accessing shared data assets.
    - Foster collaboration among users while maintaining governance.
- 💡 **Best Practices:**
    - Use principle of least privilege when setting sharing permissions.
    - Regularly audit shared assets for relevance and accuracy.
    - Tag and document published datasets for easy discovery.
- 📝 **Use Cases:**
    - Organization-wide Data Portals
    - Departmental Dashboards for Sales, Marketing, Finance
    - Data Collaboration Across Business Units

Exam Tips:

- Understand the value of centralized repositories like Analytics Hub.
- Know how access controls are enforced for shared datasets.
- Be familiar with publishing workflows and lifecycle of shared data assets.
- Expect questions on governed access and versioning of published reports.

—-Back to Index—