

Architecture Design

{ Business Strategy, planning, development and operations }

Business Requirements

Bus Process

- Stakeholder
- Change
- Team

- High Level Objectives
- Application Design considerations
- Cost considerations
- System Integration & Data Management
- Security
- Compliance & Regulations
- Success Measures

Technical Requirements

Technical Process

Functional


- Compute
- Storage
- Network

Non-functional

- Availability
- Reliability
- Scalability
- Durability

- SDLC
- ITIL
- CICD
- QA
- BCP/DR

Business Requirements

- **High Level Objectives**
 - High level vision and objectives
 - Business Use cases and Product Strategy
 - Service level objectives
- **Application Design Considerations**
 - Level of disruption tolerance
 - Implicit cost management factors
 - Total Cost of ownership (TCO)
 - Strategic or Tactical considerations
- **Cost considerations**
 - DevOps, DevSecOps
 - Managed Services, SaaS, Data Lifecycle management
- **System Integration & Data Management**
 - Integration checkpoints
 - Integration investments (RESTful APIs, point-to-point, Publisher/Subscriber)
 - Data management (how much data?, how long to store?, what to process and who to use?)
- **Security**
 - Data Confidentiality – who can see?
 - Data Integrity – who can change?
- **Compliance & Regulations**
 - HIPAA, GDPR, SOX, COPPA, PCI DSS, Open Data
 - Data Privacy regulations – PII Data
- **Success Measures**
 - Key Performance Indicators (Projects KPI, Operations KPI)
 - Return on Investment - $ROI = \frac{[value\ of\ investment - cost\ of\ investment]}{cost\ of\ investment} * 100$

Non-functional

● Availability

Availability is a measure of the time that services are functioning correctly and accessible to users. It's generally measured as a percentage of time that a system is available and responding to requests with latency not exceeding some certain threshold. SLA agreements are used as a measuring yardstick for GCP products. Ex - 99.9999 % accounts for downtime of 2.63 seconds a month. 99.999 - 26.3 seconds/month, 99.99 - 4.38 minutes/month, 99.00 - 7.31 hours/month. Following guidelines and architectural patterns can be used to ensure high availability for GCP services.

- Compute Engine - Live Migration, Managed Instance Groups, Multiple Regions and Global Load Balancing
- Kubernetes Engine - Managed Instance Groups, Replicated Master
- Storage
 - Object Storage - Fully Managed
 - File and block storage - Persistent disks support online resizing
 - Database Services
 - Self Managed DB - Shared Disk, Filesystem replication, Synchronous multi-master replication
 - Managed Databases - Fully managed (Firestore, BigQuery), Regional replication can be enabled (Cloud SQL, BigTable)
- Network
 - Redundant network connections - Dedicated or Partner interconnect
 - Premium Network Tier - Data transmitted among regions using the Google's internal network.

● Reliability

Reliability is a measure of probability that a services will continue to function under specific load over a period. Reliability is highly dependent on availability of the underlying systems and requires to consider the chances of system failures. Following guidelines and architectural patterns can be used to ensure reliability for GCP services.

- Architecture Patterns - Microservices, DevOps and distributed application framework
- Reliability Engineering
 - Service Monitoring & Alerting
 - Monitoring - Metrics, Timeseries and Dashboards
 - Logging - Centralised log management, Search and analysis process, SQL based analytics on BigQuery, Streaming consumption(Pub/Sub)
 - Alerting - Policies, Conditions and Notifications. Avoid false alerts and alert fatigue
 - Incident response procedure
 - Testing Framework - Unit test, Integration testing, System test, Reliability stress testing

Non-functional

● Scalability

Scalability is the ability of a service to adapt its infrastructure to the load on the system. It's the process of adding and removing infrastructure resources to meet workload demands efficiently. Scaling stateless applications horizontally is easy, Stateful applications are difficult to scale horizontally, and vertical scaling is often the first choice for stateful applications.

Following guidelines and architectural patterns can be used to ensure scalability for GCP services.

- Compute
 - Autoscaling can be configured based on Average CPU utilisation, Load Balancer capacity, Cloud Monitoring metrics.
 - Kubernetes Engine autoscales the number of nodes and VMs in a cluster
- Storage
 - Zonal and regional persistent disk and persistent SSDs
 - Managed databases self scale based on the workload.
- Serverless and managed services scale automatically.

● Durability

Durability is used to measure the likelihood that a stored object will be retrievable in the future.

Cloud storage has 11 9's (99.999999999%) durability guarantees, which means it's extremely unlikely that an object stored in Storage will be lost.

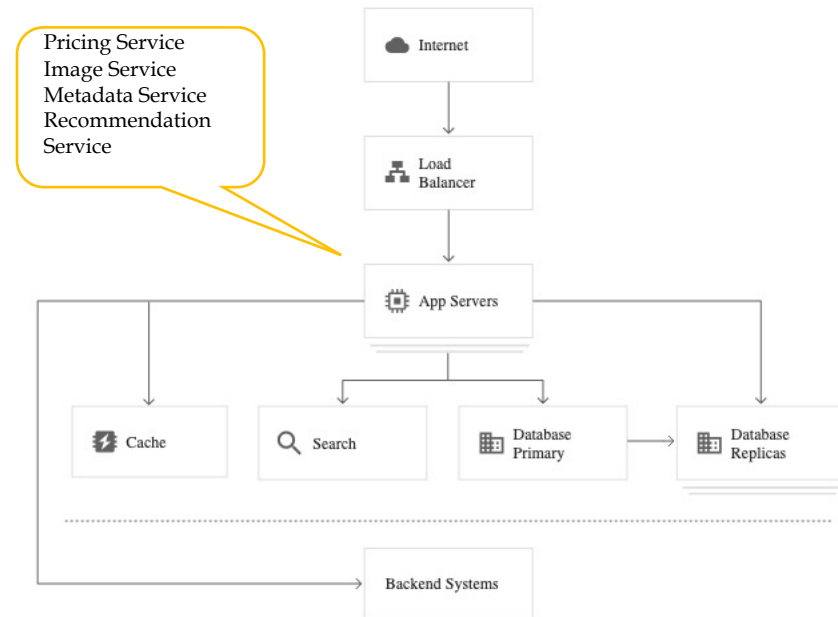
Failure domain:

- Machine-level – Hardware failure for individual machines.
- Zonal – Entire zone becomes unavailable because of building fire, power outage, fibre-optic cable loss and network isolations.
- Regional – All zones within a region become unavailable. Examples are hurricanes and large-scale earthquake.

For managed services like BigQuery, data is stored in a single region but backed up in a geographically-separated region to provide resilience to regional disaster. For soft failures, data is never lost, but for hard failures (flood, terrorist attack, earthquake, hurricanes), the recent data which is not backed up yet, would be lost.

Architecture: Microservices

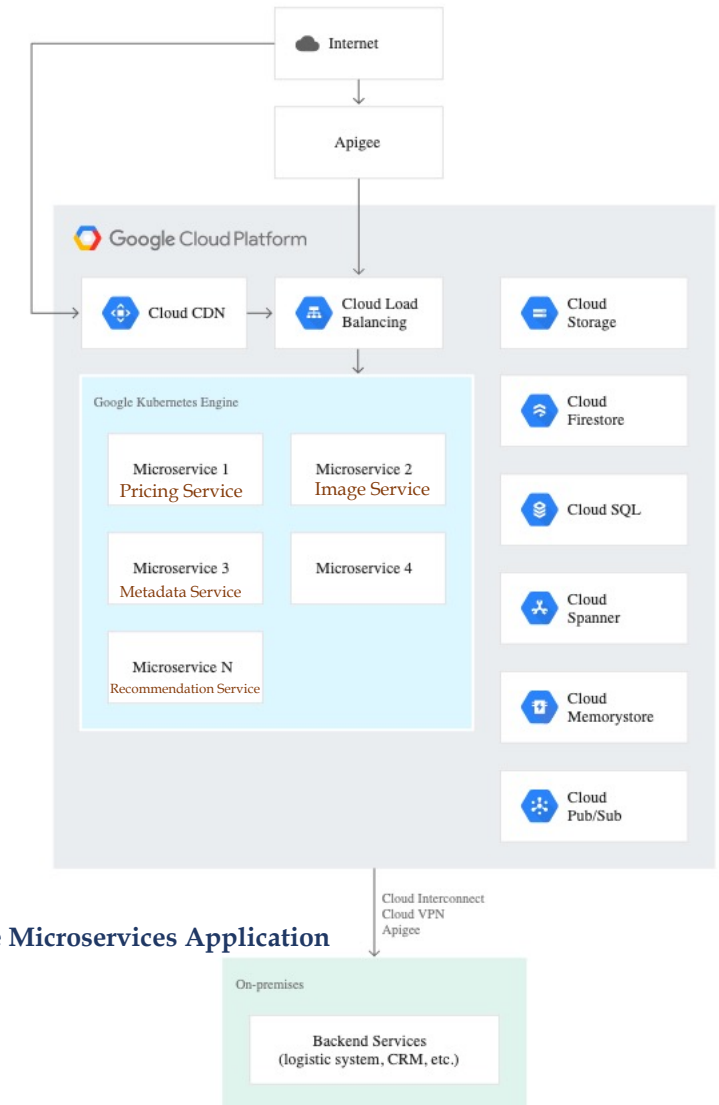
Microservices refers to an architectural style for developing applications. Microservices allow a large application to be decomposed into independent constituent parts, with each part having its own realm of responsibility. To serve a single user or API request, a microservices-based application can call many internal microservices to compose its response.



Sample Monolithic Application

- ✓ Build loosely coupled services using REST architecture
- ✓ Implement using 12-factor app development practices
- ✓ Architect stateful and stateless services to optimise scalability and reliability

- Complex inter-service dependency and traffic management
- Stateful applications can cause latency and data inconsistency
- Multiple deployments can cause version incompatibility and regression testing lifecycle



Sample Microservices Application

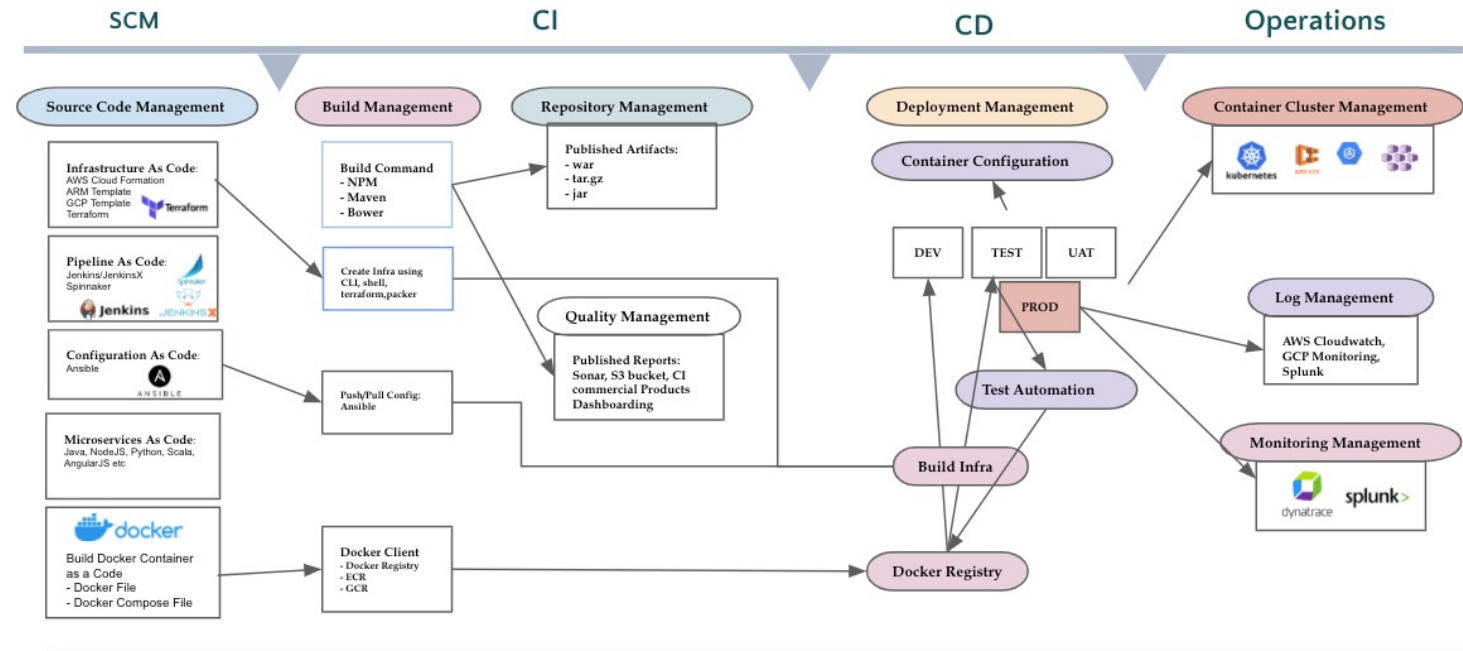
❖ Well architected microservices design :- Service boundary around stateful and stateless services.

Architecture: 12 factors app

The Twelve Factors

1. [Codebase](#)
One codebase tracked in revision control, many deploys. ex- Git
2. [Dependencies](#)
Explicitly declare and isolate dependencies. ex- npm, pip, maven etc
3. [Config](#)
Store config in the environment
4. [Backing services](#)
Treat backing services as attached resources. ex – url access for storage services.
5. [Build, release, run](#)
Strictly separate build and run stages
6. [Processes](#)
Execute the app as one or more stateless processes
7. [Port binding](#)
Export services via port binding
8. [Concurrency](#)
Scale out via the process model
9. [Disposability](#)
Maximize robustness with fast startup and graceful shutdown
10. [Dev/prod parity](#)
Keep development, staging, and production as similar as possible
11. [Logs](#)
Treat logs as event streams
12. [Admin processes](#)
Run admin/management tasks as one-off processes. Admin tasks shouldn't be a part of the application.

Twelve Factor Cloud Pipeline



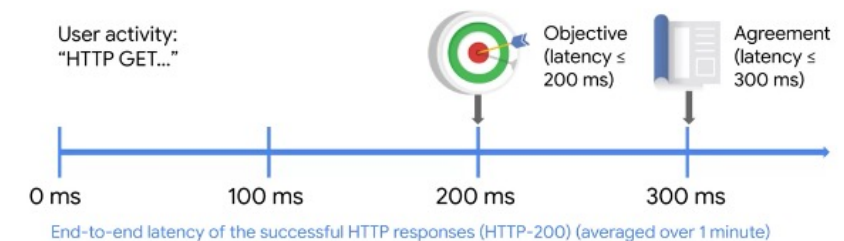
Quantitative requirements are the artefacts that can be measurable for service level outcomes.. The type of system being evaluated determines the data that can be measured,

- For user-facing systems
 - Was a request responded to, which refers to availability.
 - How long did it take to respond, which refers to latency.
 - How many requests can be handled, which refers to throughput.
- For data storage systems
 - how long does it take to read and write data? That's latency.
 - Is the data there when we need it, that's availability.
 - If there is a failure then do, we lose any data, that's durability.

SLIs, SLOs, and SLAs

- ❖ A Service Level Indicator (SLI) is a quantitative measure of some aspect of the level of service that is being provided. It is a metric, not a target.
- ❖ Service level objectives (SLOs) specify a target level for the reliability of your service. Because SLOs are key to making data-driven decisions about reliability, they're at the core of SRE practices
- ❖ Service level agreements (SLAs) are an explicit or implicit contract with users that includes consequences of meeting (or missing) the SLOs they contain.

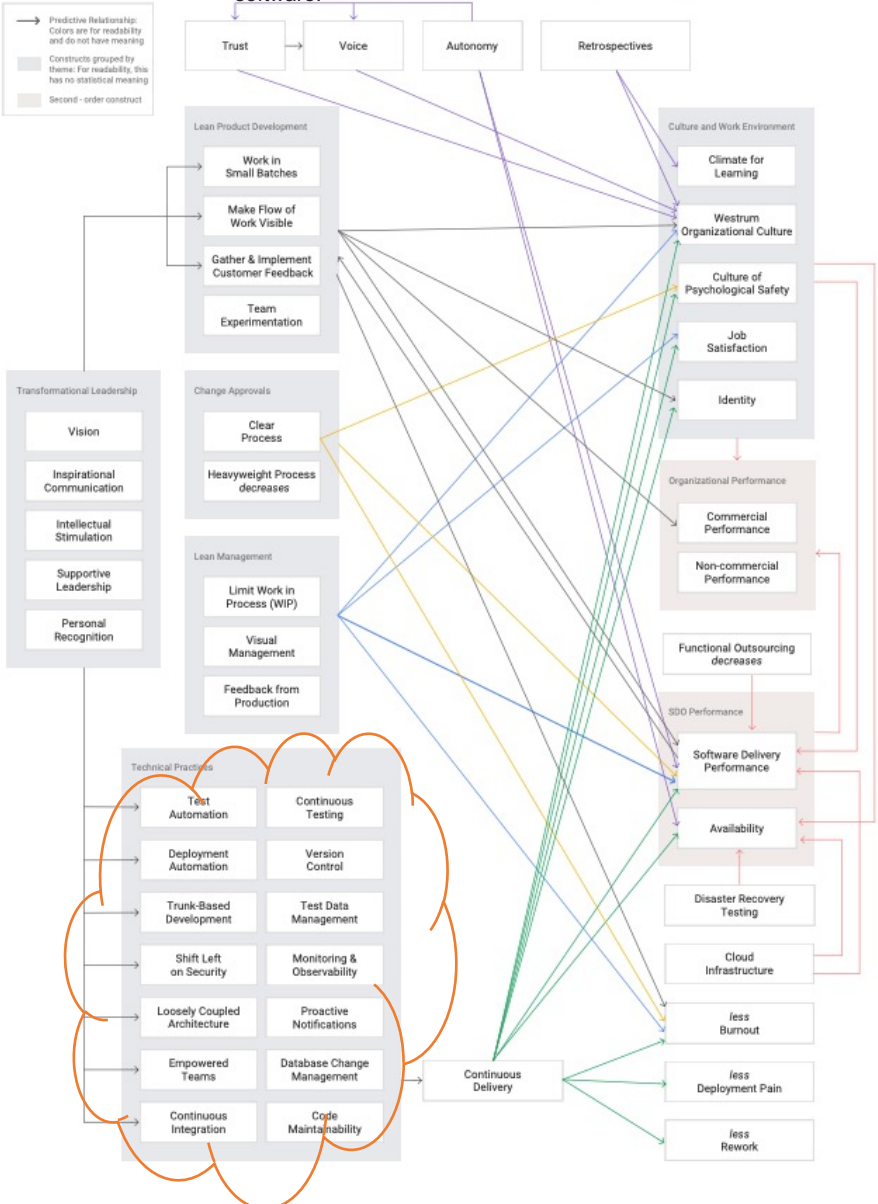
Example: SLI, SLO, and SLA



- SLI: The latency of successful HTTP responses (HTTP-200).
- SLO: The latency of 99% of the responses must be ≤ 200 ms.
- SLA: The user is compensated if 99th percentile latency exceeds 300 ms.

DORA's research program:

DORA's State of DevOps research program represents an independent view into the practices and capabilities that drive high performance in technology delivery and ultimately organizational outcomes. The research uses behavioral science to identify the most effective and efficient ways to develop and deliver software.



Google CI/CD products:



Cloud Build

Cloud Build is a service that executes builds on Google Cloud Platform infrastructure. Cloud Build can import source code from Google Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to specifications, and produce artifacts such as Docker containers or Java archives.



Artifact Registry

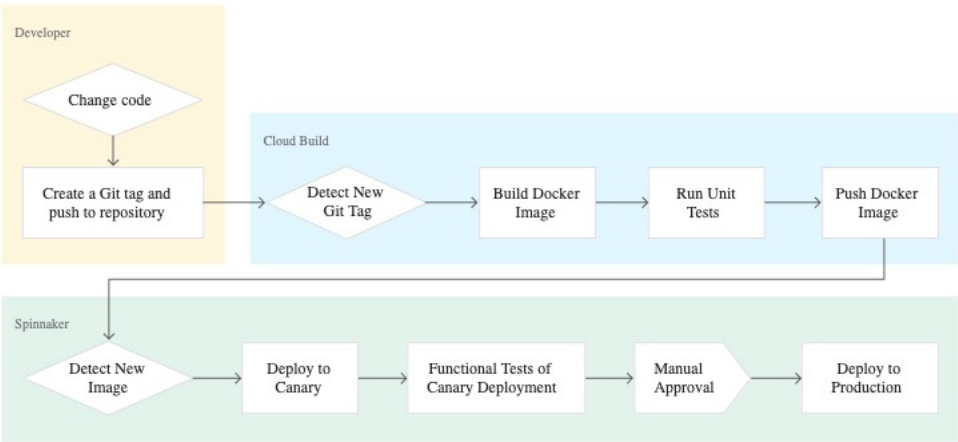
Artifact Registry provides a single location for managing packages and Docker container images. It integrates with CI/CD tools and Google Cloud runtime environments to manage the full artifact lifecycle.



Cloud Source Repositories

Cloud Source Repositories are private Git repositories hosted on Google Cloud Platform that feature fast, indexed code search across all repositories.

Sample App delivery pipeline:



Google Cloud

Google Cloud's Architecture Framework describes best practices, makes implementation recommendations and helps design the cloud deployments for business needs.

● GCP System Design Considerations

This section explains specific Google Cloud features and services that can be combined in different ways to optimize deployment for business needs.

- Geographic zones and regions based on regulatory requirements.
- Resource management based on organisation role and governance conventions.
- Identity and access management based on identity federation, MFA and audit controls
- Compute services as IaaS, PaaS and serverless capabilities
- Network resources based on internal and external networking bandwidth, security rules, trust boundaries etc.
- Storage services based on data lifecycle, regional or multiregional location, public hosting content etc.
- Database services based on structured or unstructured, latency, replication, consistency and managing capability.
- Analytics services based on ETL requirements, jobs SLA, data growth expectancy and analytics KPIs.

● Operational excellence

This section explores how operational excellence results from efficiently running, managing, and monitoring systems that deliver business value

Use these strategies to achieve operational excellence:

- ✓ **Automate build, test, and deploy.** Use continuous integration and continuous deployment (CI/CD) pipelines to build automated testing into your releases. Perform automated integration testing and deployment.
- ✓ **Monitor business objectives metrics.** Define, measure, and alert on relevant business metrics.
- ✓ **Conduct disaster recovery testing.** Don't wait for a disaster to strike. Instead, periodically verify that your disaster recovery procedures work, and test the processes regularly.

Design for Disaster Recovery:

- Define RTO (Recovery Time Objective) and RPO (Recovery Point Objective) objectives.
- Design DR plan based on the solutions for data and applications.
- Test your DR plan manually at least once a year.
- Evaluate implementing controlled fault injection to catch regressions early.
- Leverage chaos engineering to find areas of risk.

Google Cloud

Google Cloud's Architecture Framework describes best practices, makes implementation recommendations and helps design the cloud deployments for business needs.

Security, privacy and compliance

This section discusses how to plan security controls, approach privacy, and how to work with Google Cloud compliance levels.

Use these strategies to help achieve security, privacy, and compliance.

- ✓ **Implement least privilege with identity and authorization controls.** Use centralized identity management to implement the principle of least privilege and to set appropriate authorization levels and access policies.
- ✓ **Build a layered security approach.** Implement security at each level in your application and infrastructure, applying a defence-in-depth approach. Use the features in each product to limit access. Use encryption.
- ✓ **Automate deployment of sensitive tasks.** Take humans out of the workstream by automating deployment and other admin tasks.
- ✓ **Implement security monitoring.** Use automated tools to monitor your application and infrastructure. Use automated scanning in your continuous integration and continuous deployment (CI/CD) pipelines, to scan your infrastructure for vulnerabilities and to detect security incidents.

Reliability

This section describes how to apply technical and procedural requirements to architect and operate reliable services on Google Cloud.

Use these strategies to achieve reliability:

- ✓ **Reliability is defined by the user.** For user-facing workloads, measure UX metrics. For batch and streaming workloads, measure job KPIs
- ✓ **Create redundancy, include horizontal scaling, ensure overload tolerance and prevent traffic spikes.**
- ✓ **Test failure recovery and detect failure, Make incremental changes.**
- ✓ **Create, document and automate emergency response**
- ✓ **Reduce toil** - Continually aim to reduce or eliminate toil. Otherwise, operational work will eventually overwhelm operators, leaving little room for growth.

Performance and cost optimisation

This section discusses how to balance performance and cost optimizations in the deployments.

Strategies:

- ✓ **Evaluate performance requirements.** Determine the priority of your various applications and what minimum performance you require of them.
- ✓ **Use scalable design patterns.** Improve scalability and performance with autoscaling, compute choices, and storage configurations.
- ✓ **Identify and implement cost-saving approaches.** Evaluate cost for each running service while associating priority to optimize for service availability and cost. Use Export Billing to BigQuery feature, Sustained use discounts, committed use discounts, Preemptible VMs etc.