Account structure

- User identity provided by either gmail, Cloud Identity or Workspace
- Groups (to be used as role groups) provided by Cloud Identity or Workspace
- Billing account
- Projects
- IAM (Identity and Access Management)
- Default service accounts (automatically created)
- Custom service accounts (created using IAM)
- Organization policy
Account structure

- User identity provided by either gmail, Cloud Identity or Workspace
- Groups (to be used as role groups) provided by Cloud Identity or Workspace
- Billing account
- Projects
- IAM (Identity and Access Management)
- Default service accounts (automatically created)
- Custom service accounts (created using IAM)
- Organization policy

Account structure

- User identity provided by either gmail, Cloud Identity or Workspace
- Groups (to be used as role groups) provided by Cloud Identity or Workspace
- Billing account
- Projects
- IAM (Identity and Access Management)
- Default service accounts (automatically created)
- Custom service accounts (created using IAM)
- Organization policy
- **Generic best practices:**
  - Control access to resources with projects and IAM
  - Monitor and audit logs regularly
  - Avoid using the default service accounts
  - Manage users in groups

Cloud Storage

- Access granted at levels org, folder, project or bucket:
  - Permissions (specific activities allowed) propagate from top to down
  - Granted at higher level => cannot remove at lower level
  - Roles added to accounts at the project or bucket level
  - Role is a collection of permissions

- Access granted at bucket objects level:
  - Cloud Storage access control lists (ACLs)

- Be cautious when making something public
  - Used for public web content

- Best practices:
  - Don’t use PII in bucket names
  - Don’t use PII in object names (because URLs)
  - Set default object ACLs on buckets (saves time later)
  - Use signed URLs for accountless user access
  - Don’t allow buckets to be publicly writable
  - Use lifecycle rules to remove not needed sensitive data

BigQuery

- **Permissions granted to BigQuery datasets**
  - Cannot grant at the table, row or column level
  - Relevant IAM roles: BigQuery Admin, BigQuery Data Editor, BigQuery Data Owner, BigQuery Data Viewer, BigQuery Job User, BigQuery User

- **Authorized Views**
  - Used to provide row or column level permissions
  - Placed in a separate dataset with permissions different from the dataset with underlying data
  - In the dataset with the underlying data, give the view access to the data – authorized view.

- **Best practices:**
  - Use IAM roles to separate who can create and manage vs who can process the data
    - Be careful when giving all authenticated users access to data.
  - Use authorized views to restrict access to sensitive data
    - Least access principle
  - Use expiration settings to remove unneeded tables and partitions
Application Security

- Developer aims for features and functionality -> security often neglected
  - Applications - most common target of attackers
- Common application vulnerabilities:
  - Injection: SQL, LDAP, HTML
  - Cross-site scripting (XSS)
  - Weak authentication and access control
  - Sensitive data exposure
  - Security misconfiguration
  - Using components with known vulnerabilities
Application Security: Web Security Scanner

- XSS, Flash injection, mixed content (HTTP[S]), clear text passwords (lack of X-Content-Type-Options: nosniff header), insecure JS libraries
- Crawls links, activates controls and inputs, logs in with specified credentials, user-agent and QPS configurability, scheduled vs manually initiated
- Scanning large apps can take hours to complete
- Considerations: generates real load against app, can change state data in app
  - Run in test environment, use test accounts, block specific UI elements and URLs, use backup
Application Security: Identity-Aware Proxy (IAP)

- User identity verification, app access control
- Admin simplicity
  - Deploys in minutes
  - No VPN hassle
  - => saves time

Application Security: Secret Manager

- Secure way to store sensitive information
- Use to store auth credentials for apps
  - For example: API keys, passwords, certificates
- Features
  - Global names and replication
  - Versioning
  - Follows least privilege principles
  - Audit logging
  - Strong encryption
- Use IAM to grant roles/permissions
Kubernetes/GKE

- Containers, pods, nodes (VMs), clusters
- Control plane
  - Controls the other nodes in the cluster
  - Not visible in the console
  - GKE manages the control plane
- Secrets
  - Contain sensitive data (passwords, OAuth tokens, SSH keys)
  - Can be encrypted
  - Used by pods
  - Maintained separately from Google Cloud secrets
- Supported authentication for two types of accounts:
  - User accounts – known to Kubernetes, but created and managed outside (Google accounts)
  - Service accounts – created and managed by Kubernetes, but can only be used by Kubernetes created entities such as pods. Managed through Kubernetes API. Not to be confused with Google service accounts. Part of the cluster, generally used within that cluster.
- Built-in logging and monitoring
- Best practices:
  - Upgrade your GKE infrastructure in a timely fashion
  - Monitor Cluster configurations
  - Restrict direct access to control planes and nodes
  - Consider using Group Authentication
  - Use Shielded GKE nodes (same as shielded VM-s)
  - Restrict traffic between nodes
  - Use secret management (Kubernetes secrets)
  - Enable Workload Identity (simplifies authentication for accessing services external to Kubernetes)
  - Use “least privileged” Google service accounts (each node gets one, default has too broad access – create custom instead)
  - Restrict cluster discovery permissions
  - Restrict access to cluster resources
Serverless or fully managed services (no-ops)

Assumes stateless code:

- Cloud Functions
  - Types: HTTP functions and event-driven functions (background functions or CloudEvent functions).
  - Languages: Go, Python, Java, PHP, Node.js, Ruby, .NET
- App Engine Standard
  - Web based (HTTP/s) apps
  - Languages: Go, Python, Java, PHP, Node.js, Ruby
- App Engine Flexible or Cloud Run
  - Web based (HTTP/s) apps
  - Languages: anything deployable inside Docker container
- GKE
  - Any apps with specific OS or network protocols beyond HTTP/s.
  - Expandable into hybrid and multi-cloud (facilitated by Anthos).